EnergySOAR Documentation

Release latest

Sep 21, 2021

Contents

CHAPTER 1

Overview

1.1 About

Energy SOAR will make your life not only easier but also safer. By connecting with security tools and by analyzing IP, URL, files and others elements, Energy SOAR will take significant place in your imagination about working in IT Security business.

View more: https://energysoar.com

1.2 Components

CHAPTER 2

Energy SOAR installation guide

2.1 Install

Run as root in installation package directory

```
# ./install.sh -i
```

For a minimal architecture install

- TheHive
- Cortex
- Elasticsearch 7

Example installation

```
====> Do You wish to install the ENERGY SOAR TheHive, as well as the other TheHive␣
↪dependencies? [y/n] y
[..]
====> Do You wish to install the ENERGY SOAR Cortex, as well as the other Cortex␣
↪dependencies? [y/n] y
[..]
====> Do You wish to install the Elasticsearch 7? [y/n] y
[..]
====> Do You wish to initialize Cortex data? [y/n] y
[..]
====> Do You wish to initialize TheHive data? [y/n] y
[..]
```

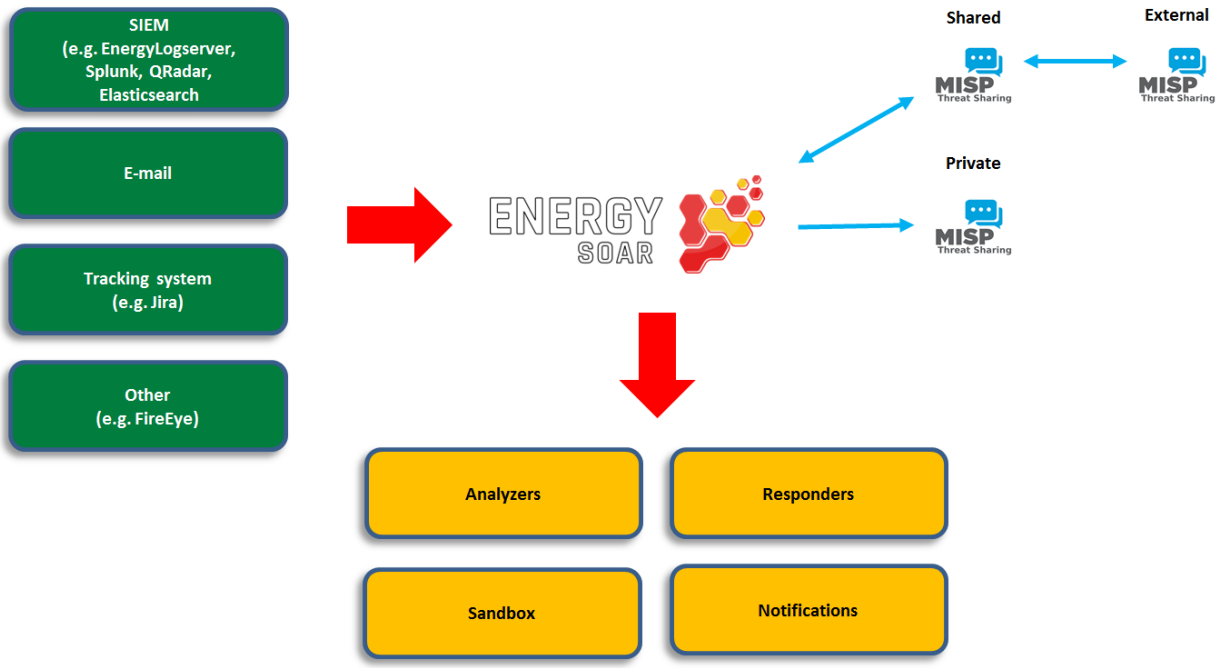
Initialize Cortex data is needed to integrate with TheHive. During this step is created api user and configured in TheHive configuration.

Initialize TheHive data create sample users and test case/alert create.

Sample users

CHAPTER 3

Architecture



CHAPTER 4

Configuration

4.1 Cortex

As described in the section above, Analyzers can only be configured using the Web interface and their associated configuration is stored in the underlying Elasticsearch database. However, the Cortex appplication configuration is stored in the `/etc/cortex/application.conf` file.

4.1.1 Database

Cortex relies on the Elasticsearch 5.x (Cortex 3 also supports Elasticsearch 6.x) search engine to store all persistent data. Elasticsearch is not part of the Cortex package. It must be installed and configured as a standalone instance which can be located on the same machine. For more information on how to set up Elasticsearch, please refer to Elasticsearch installation guide.

Three settings are required to connect to Elasticsearch:

- the base name of the index
- the name of the cluster
- the address(es) and port(s) of the Elasticsearch instance

The default settings are:

```
### Elasticsearch
search {
  # Name of the index
  index = cortex
  # Name of the Elasticsearch cluster
  cluster = hive
  # Address of the Elasticsearch instance
  host = ["127.0.0.1:9300"]
  # Scroll keepalive
  keepalive = 1m
```

(continues on next page)

(continued from previous page)

```
  # Size of the page for scroll
  pagesize = 50
  # Number of shards
  nbshards = 5
  # Number of replicas
  nbreplicas = 1
  # Arbitrary settings
  settings {
    # Maximum number of nested fields
    mapping.nested_fields.limit = 100
  }

  ### XPack SSL configuration
  # Username for XPack authentication
  #user = ""
  # Password for XPack authentication
  #password =  ""
  # Enable SSL to connect to ElasticSearch
  ssl.enabled = false
  # Path to certificate authority file
  #ssl.ca = ""
  # Path to certificate file
  #ssl.certificate = ""
  # Path to key file
  #ssl.key = ""

  ### SearchGuard configuration
  # Path to JKS file containing client certificate
  #guard.keyStore.path = ""
  # Password of the keystore
  #guard.keyStore.password = ""
  # Path to JKS file containing certificate authorities
  #guard.trustStore.path = ""
  ## Password of the truststore
  #guard.trustStore.password = ""
  # Enforce hostname verification
  #guard.hostVerification = ""
  # If hostname verification is enabled specify if hostname should be resolved
  #guard.hostVerificationResolveHostname = ""
}
```

If you use a different configuration, please make sure to modify the parameters accordingly in the `application.conf` file.

If multiple Elasticsearch nodes are used as a cluster, addresses of the master nodes must be used for the `search.host` setting. All cluster nodes must use the same cluster name:

```
search {
    host = ["node1:9300", "node2:9300"]
   ...
```

Cortex uses the TCP transport port (9300/tcp by default). Cortex cannot use the HTTP transport as of this writing (9200/tcp).

Cortex creates specific index schema (mapping) versions in Elasticsearch. Version numbers are appended to the index base name (the 8th version of the schema uses the index `cortex_8` if `search.index = cortex`). When too many documents are requested, it uses the scroll feature: the results are retrieved through pagination. You can specify

the size of the page (`search.pagesize`) and how long pages are kept in Elasticsearch (`search.keepalive`) before purging.

XPack and SearchGuard are optional and exclusive. If Cortex finds a valid configuration for XPack, SearchGuard configuration is ignored.

4.1.2 Analyzers and Responders

Cortex is able to run workers (analyzers and responders) installed locally or available as Docker image. Settings `analyzer.urls` and in `responder.urls` list paths or urls where Cortex looks for analyzers and responders. Theses settings accept:

1. a path to a directory that Cortex scans to locate workers
2. a path or an URL to a JSON file containing a JSON array of worker definitions

Worker definition is a JSON object that describe the worker, how to configure it and how to run it. If it contains a field "command", worker can be run using process runner (i.e. the command is executed). If it contains a field "dockerImage", worker can be run using docker runner (i.e. a container based on this image is started). If it contains both, the runner is chosen according to `job.runners` settings (`[docker, process]` by default).

For security reason, if worker definitions fetched from remote url (http/https) contain command, they are ignored.

You can control the number of simultaneous jobs that Cortex executes in parallel using the `analyzer.fork-join-executor` configuration item. The value depends on the number of CPU cores (`parallelism-factor` * nbCores), with a minimum (`parallelism-min`) and a maximum (`parallelism-max`).

Similar settings can also be applied to responders.

```
analyzer {
  # Directory that holds analyzers
  urls = [
    "/path/to/default/analyzers",
    "/path/to/my/own/analyzers"
  ]

  fork-join-executor {
    # Min number of threads available for analyze
    parallelism-min = 2
    # Parallelism (threads) ... ceil(available processors * factor)
    parallelism-factor = 2.0
    # Max number of threads available for analyze
    parallelism-max = 4
  }
}

responder {
  # Directory that holds responders
  urls = [
    "/path/to/default/responders",
    "/path/to/my/own/responders"
  ]

  fork-join-executor {
    # Min number of threads available for analyze
    parallelism-min = 2
    # Parallelism (threads) ... ceil(available processors * factor)
```

(continues on next page)

(continued from previous page)

```
    parallelism-factor = 2.0
    # Max number of threads available for analyze
    parallelism-max = 4
  }
}
```

4.1.3 Authentication

Like TheHive, Cortex supports local, LDAP, Active Directory (AD), X.509 SSO and/or API keys for authentication and OAuth2.

Please note that API keys can only be used to interact with the Cortex API (for example when TheHive is interfaced with a Cortex instance, it must use an API key to authenticate to it). API keys cannot be used to authenticate to the Web UI. By default, Cortex relies on local credentials stored in Elasticsearch.

Authentication methods are stored in the `auth.provider` parameter, which is multi-valued. When a user logs in, each authentication method is tried in order until one succeeds. If no authentication method works, an error is returned and the user cannot log in.

The default values within the configuration file are:

```
auth {
        # "provider" parameter contains authentication provider. It can be multi-
↪valued (useful for migration)
        # available auth types are:
        # services.LocalAuthSrv : passwords are stored in user entity (in␣
↪Elasticsearch). No configuration is required.
        # ad : use ActiveDirectory to authenticate users. Configuration is under
↪"auth.ad" key
        # ldap : use LDAP to authenticate users. Configuration is under "auth.ldap"␣
↪key
  # oauth2 : use OAuth/OIDC to authenticate users. Configuration is under "auth.oauth2
↪" and "auth.sso" keys
        provider = [local]

  # By default, basic authentication is disabled. You can enable it by setting
↪"method.basic" to true.
  method.basic = false

        ad {
                # The name of the Microsoft Windows domain using the DNS format. This␣
↪parameter is required.
                #domainFQDN = "mydomain.local"

    # Optionally you can specify the host names of the domain controllers. If not set,
↪ Cortex uses "domainFQDN".
    #serverNames = [ad1.mydomain.local, ad2.mydomain.local]

                # The Microsoft Windows domain name using the short format. This␣
↪parameter is required.
                #domainName = "MYDOMAIN"

                # Use SSL to connect to the domain controller(s).
                #useSSL = true
        }
```

(continues on next page)

(continued from previous page)

```
        ldap {
                # LDAP server name or address. Port can be specified (host:port).␣
↪This parameter is required.
                #serverName = "ldap.mydomain.local:389"

    # If you have multiple ldap servers, use the multi-valued settings.
    #serverNames = [ldap1.mydomain.local, ldap2.mydomain.local]

                # Use SSL to connect to directory server
                #useSSL = true

                # Account to use to bind on LDAP server. This parameter is required.
                #bindDN = "cn=cortex,ou=services,dc=mydomain,dc=local"

                # Password of the binding account. This parameter is required.
                #bindPW = "***secret*password***"

                # Base DN to search users. This parameter is required.
                #baseDN = "ou=users,dc=mydomain,dc=local"

                # Filter to search user {0} is replaced by user name. This parameter␣
↪is required.
                #filter = "(cn={0})"
        }

  oauth2 {
    # URL of the authorization server
    #clientId = "client-id"
    #clientSecret = "client-secret"
    #redirectUri = "https://my-cortex-instance.example/api/ssoLogin"
    #responseType = "code"
    #grantType = "authorization_code"

    # URL from where to get the access token
    #authorizationUrl = "https://auth-site.com/OAuth/Authorize"
    #tokenUrl = "https://auth-site.com/OAuth/Token"

    # The endpoint from which to obtain user details using the OAuth token, after␣
↪successful login
    #userUrl = "https://auth-site.com/api/User"
    #scope = ["openid profile"]
  }

  # Single-Sign On
  sso {
    # Autocreate user in database?
    #autocreate = false

    # Autoupdate its profile and roles?
    #autoupdate = false

    # Autologin user using SSO?
    #autologin = false

    # Name of mapping class from user resource to backend user ('simple' or 'group')
    #mapper = group
    #attributes {
```

(continues on next page)

(continued from previous page)

```
    #  login = "user"
    #  name = "name"
    #  groups = "groups"
    #  organization = "org"
    #}
    #defaultRoles = ["read"]
    #defaultOrganization = "csirt"
    #groups {
    #  # URL to retreive groups (leave empty if you are using OIDC)
    #  #url = "https://auth-site.com/api/Groups"
    #  # Group mappings, you can have multiple roles for each group: they are merged
    #  mappings {
    #    admin-profile-name = ["admin"]
    #    editor-profile-name = ["write"]
    #    reader-profile-name = ["read"]
    #  }
    #}

    #mapper = simple
    #attributes {
    #  login = "user"
    #  name = "name"
    #  roles = "roles"
    #  organization = "org"
    #}
    #defaultRoles = ["read"]
    #defaultOrganization = "csirt"
  }

}

### Maximum time between two requests without requesting authentication
session {
  warning = 5m
  inactivity = 1h
}
```

OAuth2/OpenID Connect

To enable authentication using OAuth2/OpenID Connect, edit the `application.conf` file and supply the values of `auth.oauth2` according to your environment. In addition, you need to supply:

- `auth.sso.attributes.login`: name of the attribute containing the OAuth2 user's login in retreived user info (mandatory)
- `auth.sso.attributes.name`: name of the attribute containing the OAuth2 user's name in retreived user info (mandatory)
- `auth.sso.attributes.groups`: name of the attribute containing the OAuth2 user's groups (mandatory using groups mappings)
- `auth.sso.attributes.roles`: name of the attribute containing the OAuth2 user's roles in retreived user info (mandatory using simple mapping)

Important notes

Authenticate the user using an external OAuth2 authenticator server. The configuration is:

- clientId (string) client ID in the OAuth2 server.
- clientSecret (string) client secret in the OAuth2 server.
- redirectUri (string) the url of TheHive AOuth2 page (. . . /api/ssoLogin).
- responseType (string) type of the response. Currently only "code" is accepted.
- grantType (string) type of the grant. Currently only "authorization_code" is accepted.
- authorizationUrl (string) the url of the OAuth2 server.
- authorizationHeader (string) prefix of the authorization header to get user info: Bearer, token, . . .
- tokenUrl (string) the token url of the OAuth2 server.
- userUrl (string) the url to get user information in OAuth2 server.
- scope (list of string) list of scope.

Example

```
auth {

  provider = [local, oauth2]

  [..]

  sso {
    autocreate: false
    autoupdate: false
    mapper: "simple"
    attributes {
      login: "login"
      name: "name"
      roles: "role"
    }
    defaultRoles: ["read", "analyze"]
    defaultOrganization: "demo"
  }
  oauth2 {
    name: oauth2
    clientId: "Client_ID"
    clientSecret: "Client_ID"
    redirectUri: "http://localhost:9001/api/ssoLogin"
    responseType: code
    grantType: "authorization_code"
    authorizationUrl: "https://github.com/login/oauth/authorize"
    authorizationHeader: "token"
    tokenUrl: "https://github.com/login/oauth/access_token"
    userUrl: "https://api.github.com/user"
    scope: ["user"]
  }

  [..]
}
```

4.1.4 Cache

Performance

In order to increase Cortex performance, a cache is configured to prevent repetitive database solicitation. Cache retention time can be configured for users and organizations (default is 5 minutes). If a user is updated, the cache is automatically invalidated.

Analyzer Results

Analyzer results (job reports) can also be cached. If an analyzer is executed against the same observable, the previous report can be returned without re-executing the analyzer. The cache is used only if the second job occurs within `cache.job` (the default is 10 minutes).

```
cache {
  job = 10 minutes
  user = 5 minutes
  organization = 5 minutes
}
```

Note: the global `cache.job` value can be overridden for each analyzer in the analyzer configuration Web dialog.

Note: it is possible to bypass the cache altogether (for example to get extra fresh results) through the API as explained in the API Guide or by setting the cache to *Custom* in the Cortex UI for each analyzer and specifying `0` as the number of minutes.

4.1.5 Streaming (a.k.a The Flow)

The user interface is automatically updated when data is changed in the back-end. To do this, the back-end sends events to all the connected front-ends. The mechanism used to notify the front-end is called long polling and its settings are:

- `refresh` : when there is no notification, close the connection after this duration (the default is 1 minute).
- `cache` : before polling a session must be created, in order to make sure no event is lost between two polls. If there is no poll during the cache setting, the session is destroyed (the default is 15 minutes).
- `nextItemMaxWait`, `globalMaxWait` : when an event occurs, it is not immediately sent to the front-ends. The back-end waits nextItemMaxWait and up to globalMaxWait in case another event can be included in the notification. This mechanism saves many HTTP requests.

The default values are:

```
### Streaming
stream.longpolling {
  # Maximum time a stream request waits for new element
  refresh = 1m
  # Lifetime of the stream session without request
  cache = 15m
  nextItemMaxWait = 500ms
  globalMaxWait = 1s
}
```

Entity Size Limit

The Play framework used by Cortex sets the HTTP body size limit to 100KB by default for textual content (json, xml, text, form data) and 10MB for file uploads. This could be too small in some cases so you may want to change it with the following settings in the `application.conf` file:

```
### Max textual content length
play.http.parser.maxMemoryBuffer=1M
### Max file size
play.http.parser.maxDiskBuffer=1G
```

Note: if you are using a NGINX reverse proxy in front of Cortex, be aware that it doesn't distinguish between text data and a file upload. So, you should also set the `client_max_body_size` parameter in your NGINX server configuration to the highest value among the two: file upload and text size as defined in Cortex `application.conf` file.

4.1.6 HTTPS

Enable HTTPS directly on Cortex is not supported anymore. You must install a reverse proxy in front of Cortex. Below an example of NGINX configuration:

```
        server {
                        listen 443 ssl;
                        server_name cortex.example.com;

                        ssl_certificate                         ssl/cortex_cert.pem;
                        ssl_certificate_key                 ssl/cortex_key.pem;

                        proxy_connect_timeout   600;
                        proxy_send_timeout      600;
                        proxy_read_timeout      600;
                        send_timeout            600;
                        client_max_body_size    2G;
                        proxy_buffering off;
                        client_header_buffer_size 8k;

                        location / {
                                        add_header
↪       Strict-Transport-Security "max-age=31536000; includeSubDomains";
                                        proxy_pass                  http://127.0.0.
↪1:9001/;
                                        proxy_http_version          1.1;
                                        proxy_set_header Connection "";
                        }
        }
```

4.2 TheHive

4.2.1 `secret.conf` file

This file contains a secret that is used to define cookies used to manage the users session. As a result, one instance of TheHive should use a unique secret key.

Example

```
## Play secret key
play.http.secret.key="dgngu325mbnbc39cxas415kb24503836y2vsvsg465989fbsvop9d09ds6df6"
```

Warrning

In the case of a cluster of Energy SOAR nodes, all nodes should have the same secret.conf file with the same secret key. The secret is used to generate user sessions.

4.2.2 Service

Listen address & port

By default the application listens on all interfaces and port 9000. This is possible to specify listen address and ports with following parameters in the application.conf file:

```
http.address=127.0.0.1
http.port=9000
```

Context

If you are using a reverse proxy, and you want to specify a location (ex: /thehive), updating the configuration of TheHive is also required

Example

play.http.context: "/thehive" Specific configuration for streams# If you are using a reverse proxy like Nginx, you might receive error popups with the following message: StreamSrv 504 Gateway Time-Out.

You need to change default setting for long polling refresh, Set stream.longPolling.refresh accordingly.

Example

```
stream.longPolling.refresh: 45 seconds
```

Manage content lengh

Content length of text and files managed by the application are limited by default.

These values are set with default parameters:

```
# Max file size
play.http.parser.maxDiskBuffer: 128MB
```

```
# Max textual content length
play.http.parser.maxMemoryBuffer: 256kB
If you feel that these should be updated, edit /etc/thehive/application.conf file and␣
↪update these parameters accordingly.
```

Tip

if you are using a NGINX reverse proxy in front of Energy SOAR, be aware that it doesn't distinguish between text data and a file upload.

So, you should also set the client_max_body_size parameter in your NGINX server configuration to the highest value among the two: file upload and text size defined in TheHive application.conf file.

4.2.3 Manage configuration files

Energy SOAR uses HOCON as configuration file format. This format gives enough flexibility to structure and organise the configuration of Energy SOAR.

TheHive is delivered with following files, in the folder `/etc/thehive`:

`logback.xml` containing the log policy

`secret.conf` containing a secret key used to create sessions. This key should be unique per instance (in the case of a cluster, this key should be the same for all nodes of this cluster) `application.conf`

HOCON file format let you organise the configuration to have separate files for each purpose. It is the possible to create a /etc/thehive/application.conf.d folder and have several files inside that will be included in the main file /etc/thehive/application.conf.

At the end, the following configuration structure is possible:

```
/etc/thehive
|-- application.conf
|-- application.conf.d
|   |-- secret.conf
|   |-- service.conf
|   |-- database.conf
|   |-- storage.conf
|   |-- cluster.conf
|   |-- authentication.conf
|   |-- cortex.conf
|   |-- misp.conf
|   `-- webhooks.conf
`-- logback.xml
```

And the content of `/etc/thehive/application.conf`:

```
## Include Play secret key
# More information on secret key at https://www.playframework.com/documentation/2.8.x/
↪ApplicationSecret
include "/etc/thehive/application.conf.d/secret.conf"

## Service
include "/etc/thehive/application.conf.d/service.conf"

## Database
include "/etc/thehive/application.conf.d/database.conf"

## Storage
include "/etc/thehive/application.conf.d/storage.conf"

## Cluster
include "/etc/thehive/application.conf.d/cluster.conf"

## Authentication
include "/etc/thehive/application.conf.d/authentication.conf"

## Cortex
include "/etc/thehive/application.conf.d/cortex.conf"

## MISP
include "/etc/thehive/application.conf.d/misp.conf"
```

(continues on next page)

(continued from previous page)

```
## Webhooks
include "/etc/thehive/application.conf.d/webhooks.conf"
```

4.3 SSL

Energy SOAR instalation script create self-signed certificates. Those certificates are stored under `/etc/thehive/ssl/` directory.

You can setup your own path in `/etc/nginx/conf.d/energysoar.conf`.

```
    ssl_certificate     /etc/thehive/ssl/nginx-selfsigned.crt;
    ssl_certificate_key /etc/thehive/ssl/nginx-selfsigned.key;
```

CHAPTER 5

User guide

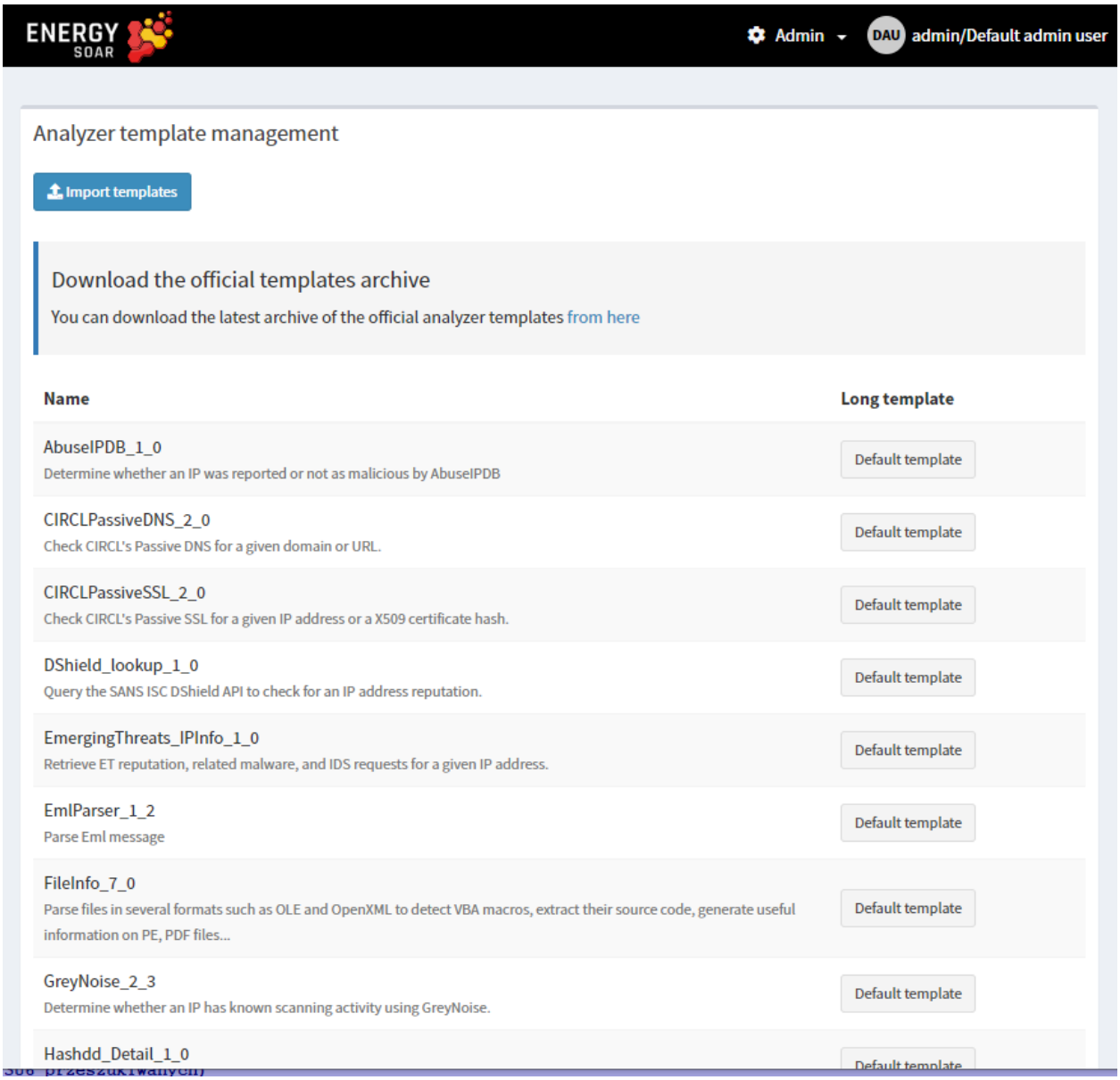
5.1 Administration

5.1.1 Manage analyzer template

TheHive will display the analysis summary the same way for all analyzers: display a tag using taxonomies and level color.

List analyzer templates

The management page is accessible from the header menu through the Admin > Analyzer templates menu and required a use with the manageAnalyzerTemplate permission (refer to Profiles and permissions).



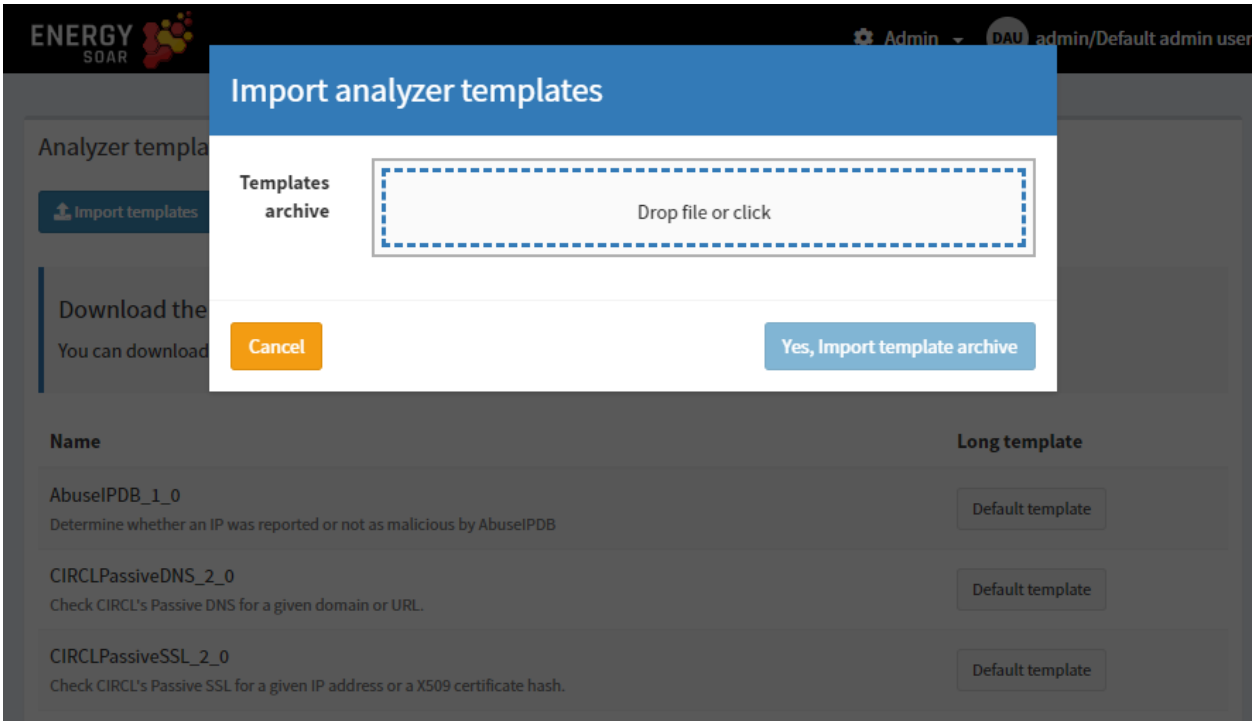
Analyzer templates are still customisable via the UI and can also be imported.

Import analyzer templates

TheHive Project provides a set of analyzer templates (we use the same report-templates.zip archive for backward compatibility reasons).

The template archive is available at https://download.thehive-project.org/report-templates.zip.

To import the zip file, click on the Import templates, this opens the import dialog. Drop the zip files or click to select it from your storage and finally click Yes, import template archive.



Note that analyzer templates are global and common to all the organisations.

5.1.2 User Profiles management

Permissions

A Profile is a set of permissions attached to a User and an Organisation. It defines what the user can do on an object hold by the organisation. TheHive has a finite list of permissions:

- manageOrganisation (1) : the user can create, update an organisation
- manageConfig (1): the user can update configuration
- manageProfile (1): the user can create, update and delete profiles
- manageTag (1): the user can create, update and delete tags
- manageCustomField (1): the user can create, update and delete custom fields
- manageCase: the user can create, update and delete cases
- manageObservable: the user can create, update and delete observables
- manageAlert: the user can create, update and import alerts
- manageUser: the user can create, update and delete users
- manageCaseTemplate: the user can create, update and delete case template
- manageTask: the user can create, update and delete tasks
- manageShare: the user can share case, task and observable with other organisation
- manageAnalyse (2): the user can execute analyse
- manageAction (2): the user can execute actions

- manageAnalyzerTemplate (2): the user can create, update and delete analyzer template (previously named report template)

(1) Organisations, configuration, profiles and tags are global objects. The related permissions are effective only on "admin" organisation. (2) Actions, analysis and template is available only if Cortex connector is enabled

NOTE

Read information doesn't require specific permission. By default, users in an organisation can see all data shared with that organisation (cf. shares, discussed in Organisations,Users and sharing).

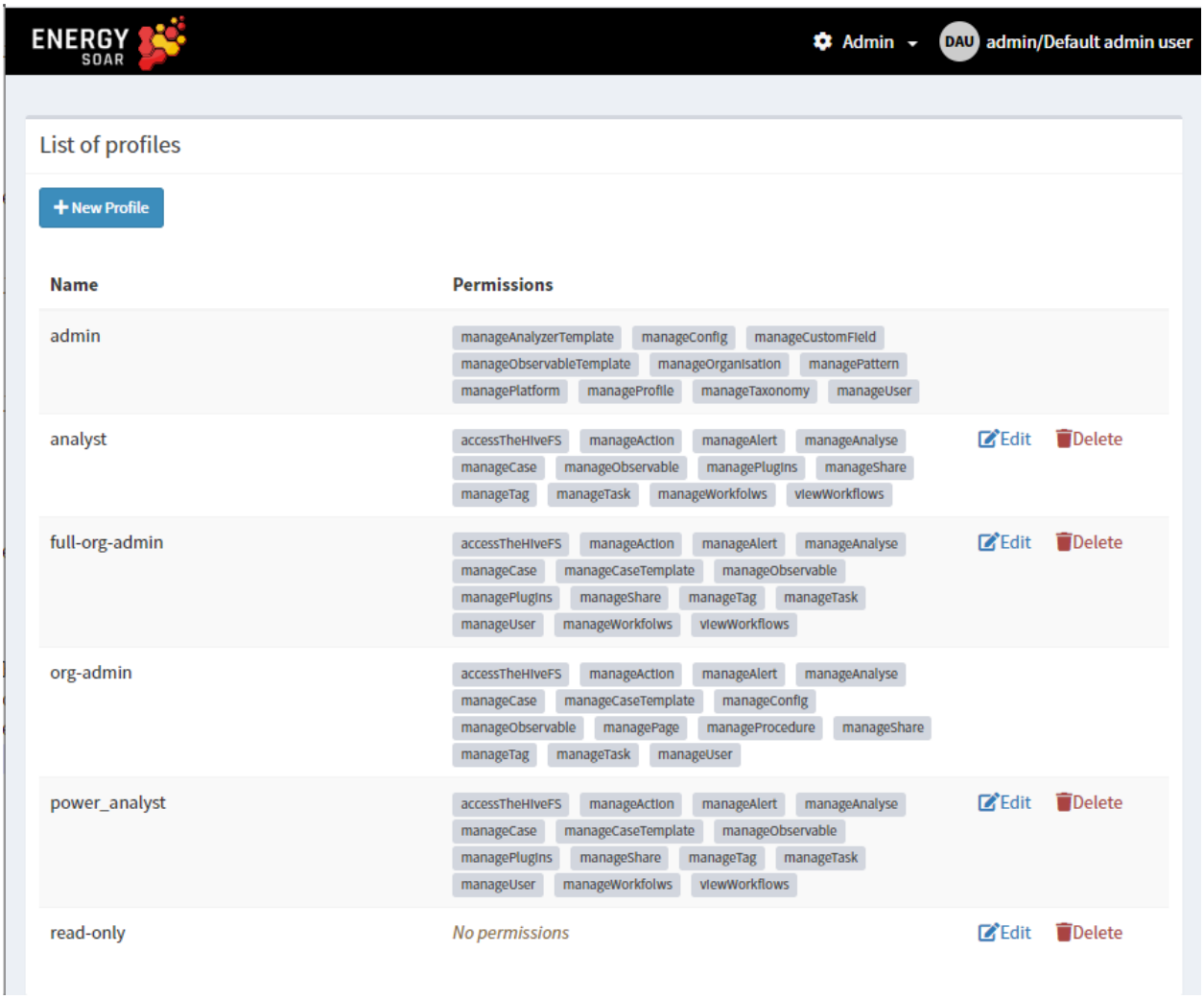
Profiles

We distinguish two types of profiles:

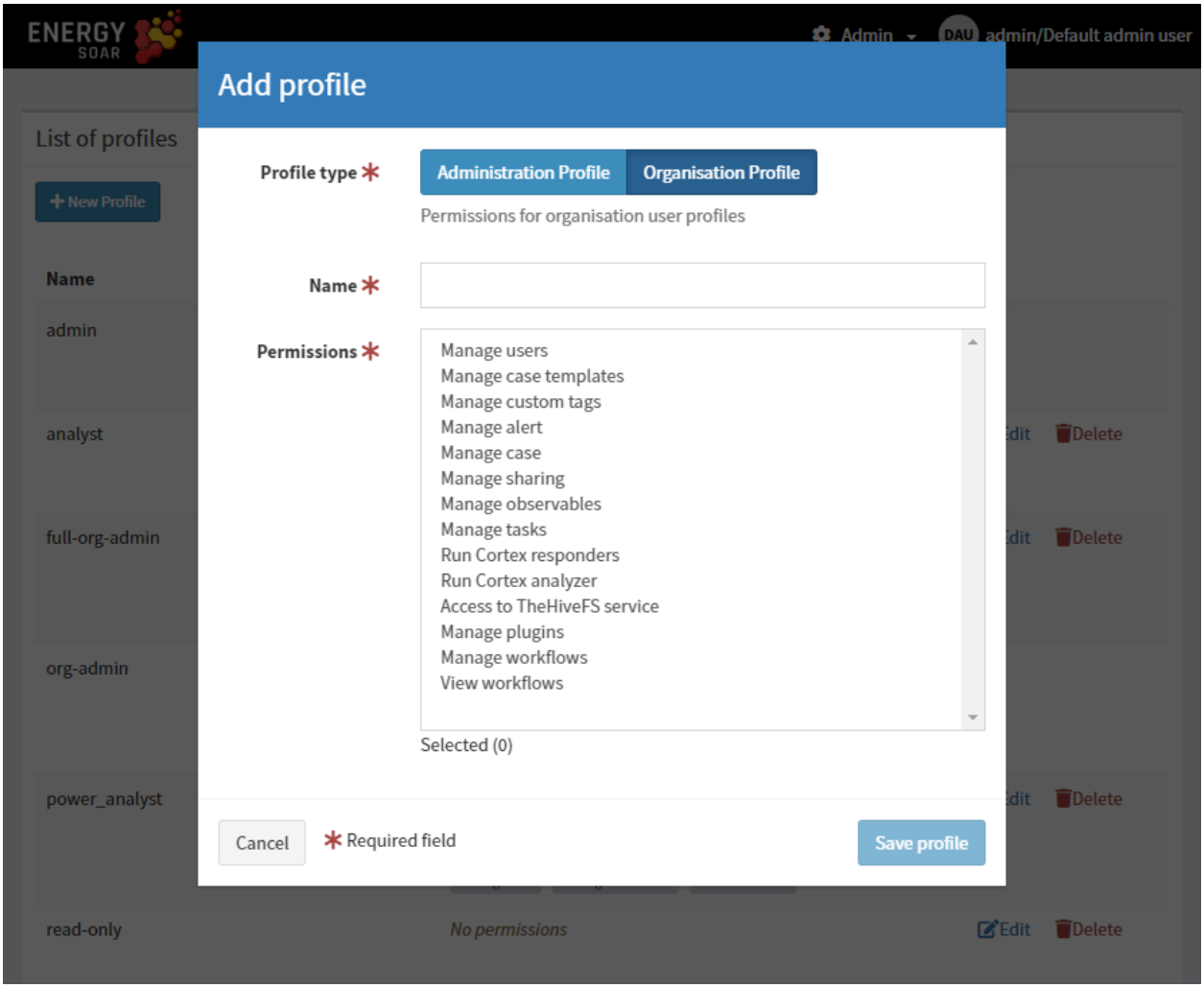
- Administration Profiles
- Organisation Profiles

The management page is accessible from the header menu through the Admin > Profiles menu and required a use with the manageProfile permission (refer to the section above).

TheHive comes with default profiles but they can be updated and removed (if not used). New profiles can be created.



Once the New Profile button is clicked, a dialog is opened asking for the profile type, a name for the profile and a selection of permissions. Multiple selection can be made using CTRL+click.



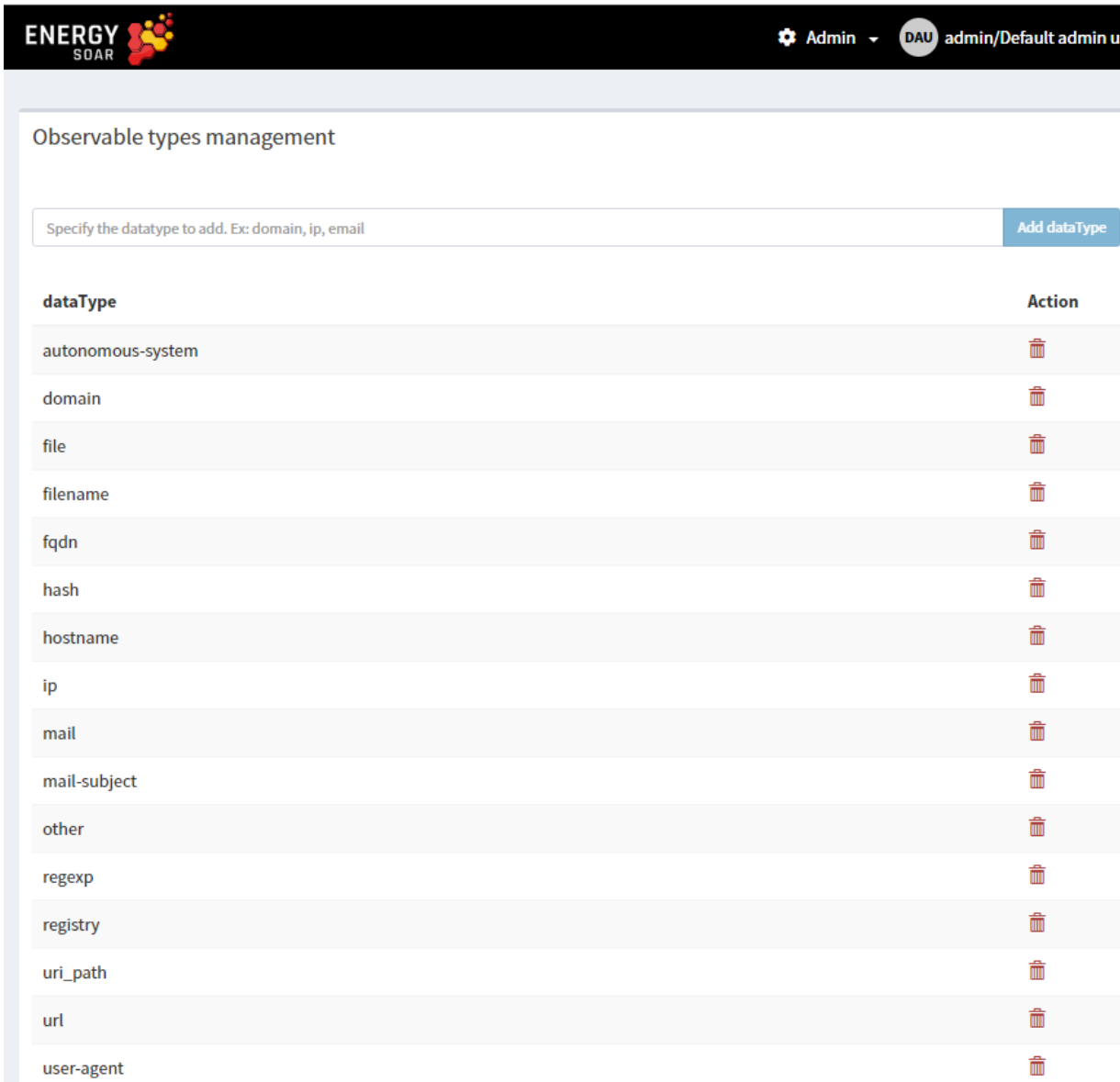
If it is used, a profile can't be remove but can be updated.

Default profiles are:

- admin: can manage all global objects and users. Can't create case.
- analyst: can manage cases and other related objects (observables, tasks, . . .), including shring them
- org-admin: all permissions except those related to global objects
- read-only: no permission

Observable types

You can edit observable types in the administrator panel.



Admin > Observable

5.2 Alerts

5.3 Responders

5.3.1 Responders list

- Symantec Endpoint Protection Unquarantine Host_0_1
- AMPforEndpoints_SCDAdd_1_0
- Crowdstrike_Falcon_Custom_IOC_API_1_0
- Symantec Messaging Gateway Unblock Email_0_1

- Symantec Messaging Gateway Block Domain_0_1
- Check Point Block IP_0_1
- SendGrid_1_0
- Check Point Unblock IP_0_1
- Redmine_Issue_1_0
- DNS-RPZ_1_0
- Info Blox Block IP_0_1
- Info Blox Block Domain_0_1
- AMPforEndpoints_MoveGUID_1_0
- RT4-CreateTicket_1_0
- Symantec Messaging Gateway Block Email_0_1
- Mailer_1_0
- KnowBe4_1_0
- Velociraptor_Flow_0_1
- DomainToolsIris_AddRiskyDNSTag_1_0
- Virustotal_Downloader_0_1
- Wazuh_1_0
- Symantec Endpoint Protection Unblock Hash_0_1
- ZEROFOX_Close_alert_1_0
- Minemeld_1_0
- Umbrella_Blacklister_1_1
- ZEROFOX_Takedown_request_1_0
- Symantec Endpoint Protection Quarantine Host_0_1
- DomainToolsIris_CheckMaliciousTags_1_0
- Symantec Messaging Gateway Unblock IP_0_1
- Symantec Messaging Gateway Block IP_0_1
- AMPforEndpoints_IsolationStart_1_0
- AMPforEndpoints_IsolationStop_1_0
- QRadar_Auto_Closing_Offense_1_0
- Symantec Endpoint Protection Block Hash_0_1
- Info Blox Delete Rule_0_1
- Symantec Messaging Gateway Unblock Domain_0_1
- AMPforEndpoints_SCDRemove_1_0

5.4 Analyzers

5.4.1 Analyzers list

- IPVoid_1_0
- OpenCTI_SearchObservable_1_0
- SEKOIAIntelligenceCenter_Indicators_1_0
- SEKOIAIntelligenceCenter_Context_1_0
- HIBP_Query_2_0
- DNSSinkhole_1_0
- DomainToolsIris_Investigate_1_0
- Cyberprotect_ThreatScore_1_0
- Autofocus_SearchJSON_1_0
- DomainTools_Reputation_2_0
- VirusTotal_GetReport_3_0
- MaxMind_GeoIP_4_0
- FireEyeiSight_1_0
- Malwares_GetReport_1_0
- Mnemonic_pDNS_Public_3_0
- DomainTools_Risk_2_0
- PassiveTotal_Osint_2_0
- CIRCLPassiveDNS_2_0
- CyberChef_FromHex_1_0
- PassiveTotal_Passive_Dns_2_1
- Shodan_Host_1_0
- DomainTools_WhoisLookupUnparsed_2_0
- PassiveTotal_Host_Pairs_2_0
- Hunterio_DomainSearch_1_0
- CyberChef_FromCharCode_1_0
- MISPWarningLists_2_0
- DomainTools_ReverseIPWhois_2_0
- AbuseIPDB_1_0
- TorProject_1_0
- CIRCLPassiveSSL_2_0
- Fortiguard_URLCategory_2_1
- Splunk_Search_User_Agent_3_0
- Yara_2_0

- EmergingThreats_DomainInfo_1_0
- DNSDB_DomainName_2_0
- PhishTank_CheckURL_2_1
- IPinfo_Hosted_Domains_1_0
- SpamhausDBL_1_0
- PassiveTotal_Trackers_2_0
- ThreatResponse_1_0
- FileInfo_7_0
- Maltiverse_Report_1_0
- BackscatterIO_GetObservations_1_0
- OTXQuery_2_0
- Investigate_Sample_1_0
- MetaDefenderCloud_Reputation_1_0
- Autofocus_SearchIOC_1_0
- Splunk_Search_Mail_Email_3_0
- LastInfoSec_1_0
- Patrowl_GetReport_1_0
- NSRL_1_0
- PhishingInitiative_Scan_1_0
- C1fApp_1_0
- RecordedFuture_risk_1_0
- Nessus_2_0
- SecurityTrails_Passive_DNS_1_0
- JoeSandbox_File_Analysis_Inet_2_0
- Virusshare_2_0
- GreyNoise_2_3
- DomainTools_ReverseIP_2_0
- Yeti_1_0
- StaxxSearch_1_0
- SinkDB_1_1
- MalwareBazaar_1_0
- Robtex_Forward_PDNS_Query_1_0
- WOT_Lookup_2_0
- Splunk_Search_Hash_3_0
- Autofocus_GetSampleAnalysis_1_0
- VirusTotal_Scan_3_0

- EmergingThreats_IPInfo_1_0
- Shodan_ReverseDNS_1_0
- Shodan_Host_History_1_0
- PassiveTotal_Whois_Details_2_0
- Urlscan_io_Search_0_1_1
- DomainTools_WhoisLookup_2_0
- PassiveTotal_Malware_2_0
- DomainTools_ReverseNameServer_2_0
- IntezerCommunity_1_0
- DNSDB_IPHistory_2_0
- GoogleSafebrowsing_2_0
- PassiveTotal_Enrichment_2_0
- PayloadSecurity_File_Analysis_1_0
- Msg_Parser_3_0
- DomainMailSPFDMARC_Analyzer_1_1
- PassiveTotal_Unique_Resolutions_2_0
- Splunk_Search_User_3_0
- CuckooSandbox_Url_Analysis_1_2
- BackscatterIO_Enrichment_1_0
- Hashdd_Detail_1_0
- DomainTools_ReverseWhois_2_0
- Threatcrowd_1_0
- CyberCrime-Tracker_1_0
- EmailRep_1_0
- URLhaus_2_0
- MISP_2_1
- TeamCymruMHR_1_0
- Hashdd_Status_1_0
- DShield_lookup_1_0
- EmergingThreats_MalwareInfo_1_0
- StopForumSpam_1_0
- DomainTools_HostingHistory_2_0
- CyberChef_FromBase64_1_0
- Abuse_Finder_3_0
- Investigate_Categorization_1_0
- SecurityTrails_Whois_1_0

- DomainTools_WhoisHistory_2_0
- MetaDefenderCloud_Scan_1_0
- PassiveTotal_Ssl_Certificate_History_2_0
- Splunk_Search_Other_3_0
- Malpedia_1_0
- MetaDefenderCore_Scan_1_0
- Splunk_Search_Registry_3_0
- Crt_sh_Transparency_Logs_1_0
- IPinfo_Details_1_0
- CERTatPassiveDNS_2_0
- Urlscan_io_Scan_0_1_0
- ProofPoint_Lookup_1_0
- PayloadSecurity_Url_Analysis_1_0
- Shodan_DNSResolve_1_0
- Splunk_Search_Mail_Subject_3_0
- GoogleDNS_resolve_1_0_0
- DomainToolsIris_Pivot_1_0
- MetaDefenderCloud_GetReport_1_0
- Hipposcore_2_0
- Shodan_InfoDomain_1_0
- CuckooSandbox_File_Analysis_Inet_1_2
- JoeSandbox_File_Analysis_Noinet_2_0
- GoogleVisionAPI_WebDetection_1_0_0
- TalosReputation_1_0
- Splunk_Search_IP_3_0
- TorBlutmagie_1_0
- SpamAssassin_1_0
- Splunk_Search_Domain_FQDN_3_0
- FireHOLBlocklists_2_0
- NERD_1_0
- ThreatGrid_1_0
- Robtex_Reverse_PDNS_Query_1_0
- PassiveTotal_Ssl_Certificate_Details_2_0
- VMRay_3_0
- DNSDB_NameHistory_2_0
- PhishingInitiative_Lookup_2_0

- SoltraEdge_1_0
- Pulsedive_GetIndicator_1_0
- IBMXForce_Lookup_1_0
- Splunk_Search_URL_URI_Path_3_0
- JoeSandbox_Url_Analysis_2_0
- Censys_1_0
- Malwares_Scan_1_0
- Robtex_IP_Query_1_0
- HippoMore_2_0
- HybridAnalysis_GetReport_1_0
- EmlParser_1_2
- ClamAV_FileInfo_1_1
- ForcepointWebsensePing_1_0
- Shodan_Search_2_0
- Umbrella_Report_1_0
- PassiveTotal_Components_2_0
- MetaDefenderCore_GetReport_1_0
- MalwareClustering_Search_1_0
- Mnemonic_pDNS_Closed_3_0
- Splunk_Search_File_Filename_3_0
- UnshortenLink_1_2
- Onyphe_Summary_1_0
- AnyRun_Sandbox_Analysis_1_0

5.5 Cases

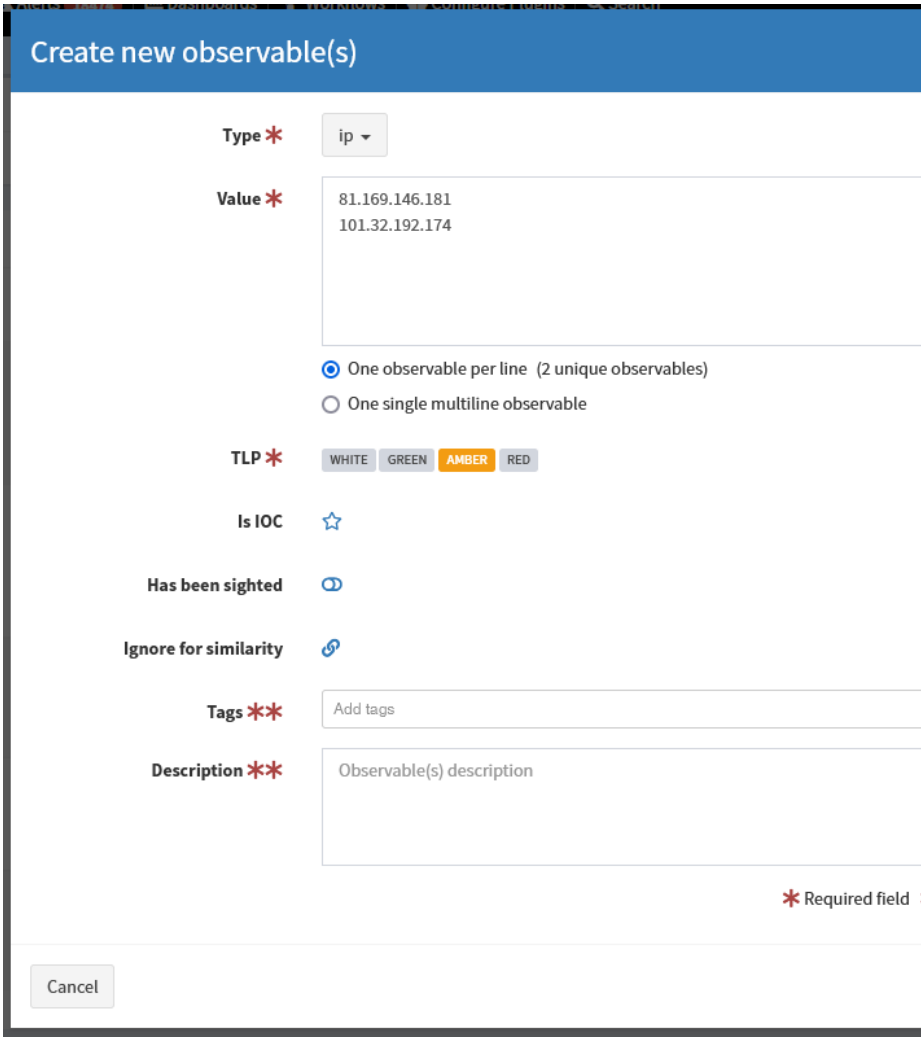
5.5.1 Observables

Observables are pieces of information added to a case.

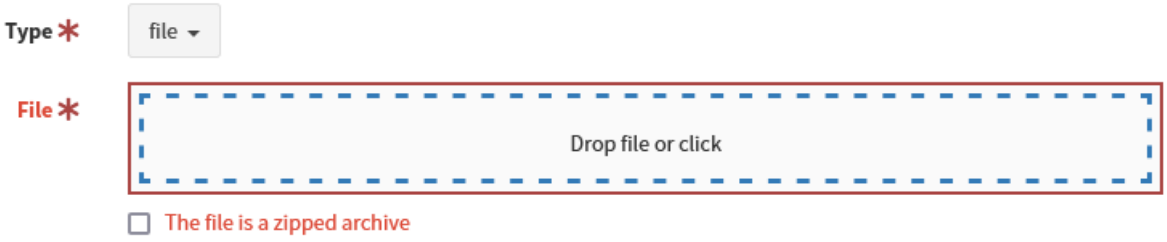
How to add observables into Case

Perform the following steps to add an observable:

1. Click Add observable(s) button: 

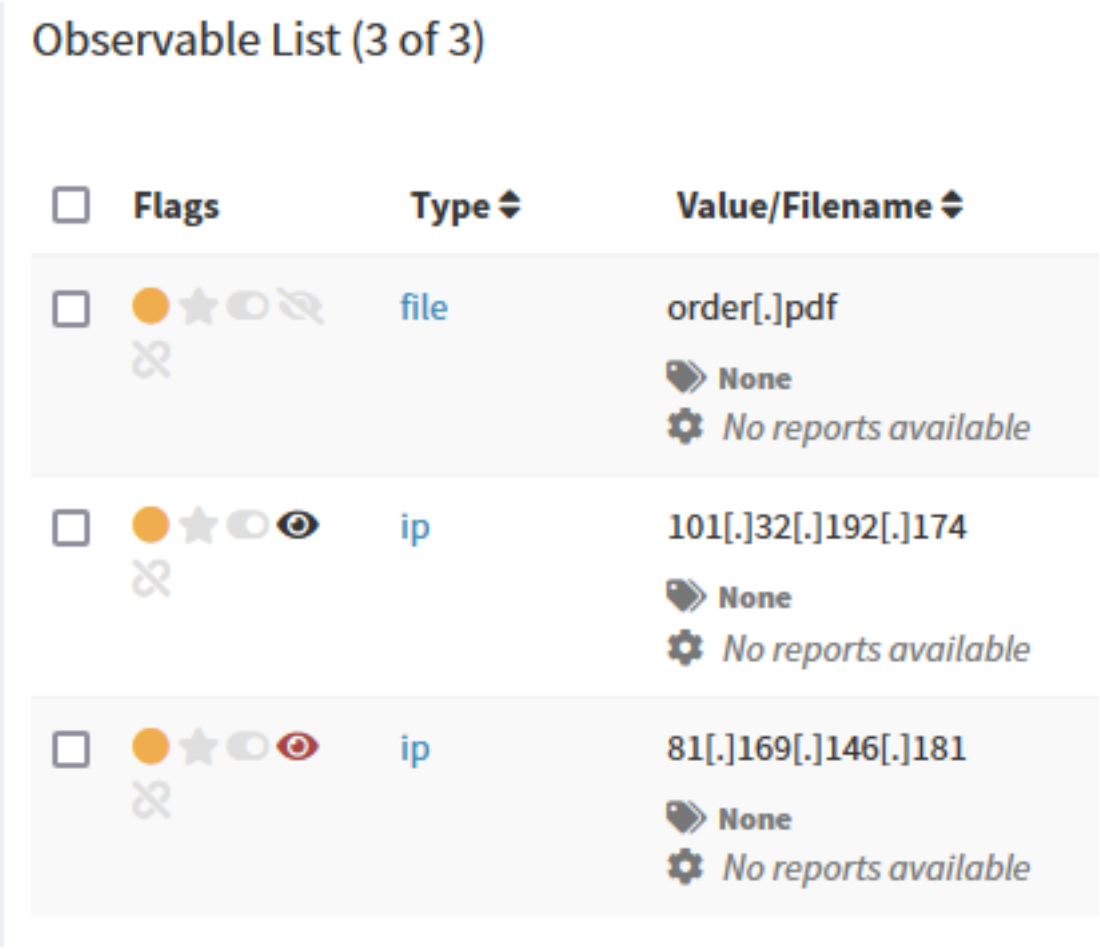


2. Create new observable(s) window appears:
3. Select type e.g. ip, domain, url, mail. If you choose file type, you can upload a file. Zipped archives are supported.

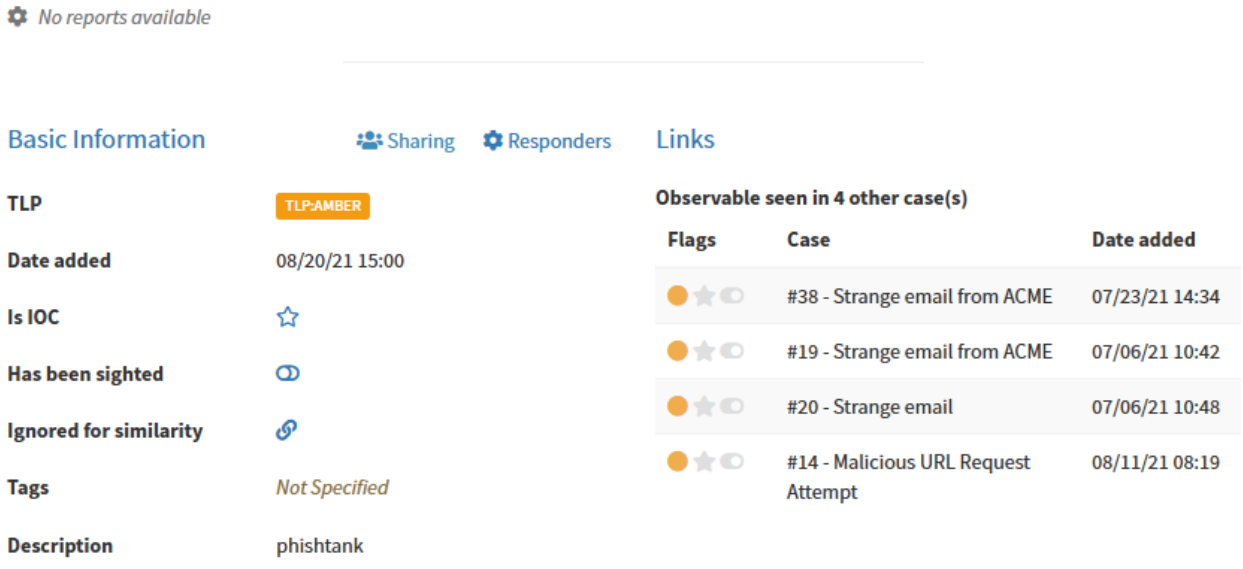


4. You can add one single observables or many observables at once - one observable per line.
5. Select appropriate TLP flag.
6. (Optional) IOC flag indicates observables classified as True Positive. Only IOC-flagged observables are exported to MISP instances.
7. (Optional) You can also set “Has been sighted” toggle to mark observables which have been seen.
8. (Optional) If you click “Ignore for similarity”, you will disable “Observable seen in other cases” list.
9. Add tags and/or description.
10. Click Create observable(s). On Observable List you can check if observables have been seen in other cases:

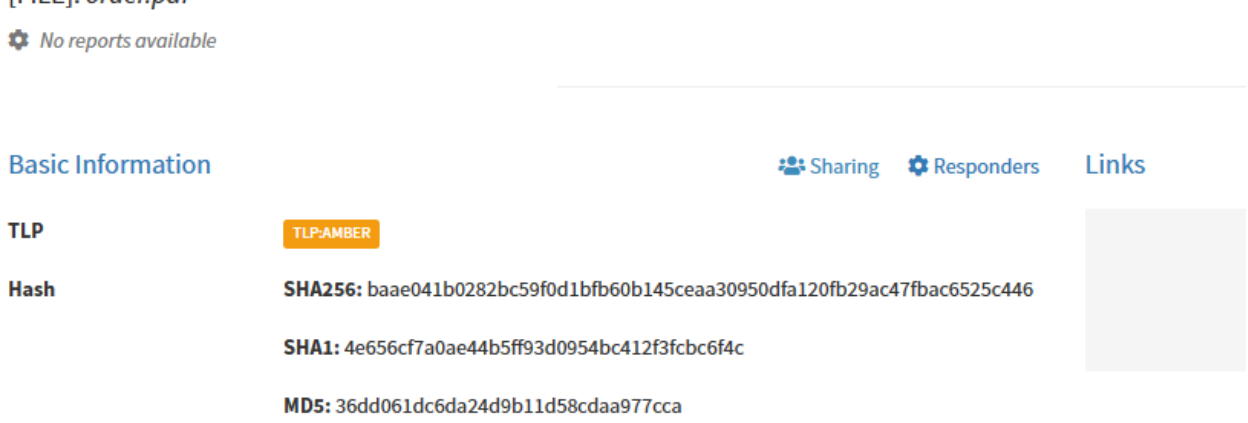
- Black eye: Observable seen in other cases,
- Red eye: Observable seen in other cases and flagged as IOC there.



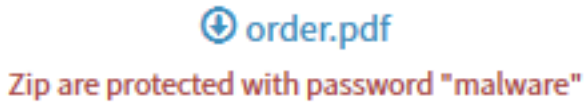
You can display details and check cases where the observable has been seen:



After uploading file-type observables hashes are automatically calculated:



If you want to download file observable, it will be zipped and password protected:



You can run various analyzers (e.g. VirusTotal, MaxMind_GeoIP) and responders (e.g. block IP, domain, e-mail) against observables.

5.6 Organisation

5.7 Reports

CHAPTER 6

Operations

CHAPTER 7

API

In this documentation we use local adresses. When you connect externaly then you should external IP under secure http - https://YOUR_IP.

7.1 Base API Guide

7.1.1 Authentication

Most API calls require authentication. Credentials can be provided using a session cookie, an API key or directly using HTTP basic authentication (when enabled).

Using API key

Session cookie is suitable for browser authentication, not for a dedicated tool. The easiest solution if you want to write a tool that leverages Base module's API is to use API key authentication. API keys can be generated using the Web interface of the product, under the user admin area. For example, to list cases, use the following curl command:

```
curl -H 'Authorization: Bearer ***API*KEY***' http://127.0.0.1:9000/base/api/case
```

Using basic authentication

Base module also supports basic authentication (disabled by default). You can enable it by adding `auth.method.basic=true` in the configuration file.

```
curl -u mylogin:mypassword http://127.0.0.1:9000/base/api/case
```

7.1.2 Alert

Model definition

Required attributes:

- `title` (text) : title of the alert
- `description` (text) : description of the alert
- `severity` (number) : severity of the alert (1: low; 2: medium; 3: high) **default=2**
- `date` (date) : date and time when the alert was raised **default=now**
- `tags` (multi-string) : case tags **default=empty**
- `tlp` (number) : TLP (`0`: `white`; `1`: `green`; `2`: `amber`; `3`: `red`) **default=2**
- `status` (AlertStatus) : status of the alert (*New*, *Updated*, *Ignored*, *Imported*) **default=New**
- `type` (string) : type of the alert (read only)
- `source` (string) : source of the alert (read only)
- `sourceRef` (string) : source reference of the alert (read only)
- `artifacts` (multi-artifact) : artifact of the alert. It is a array of JSON object containing artifact attributes **default=empty**
- `follow` (boolean) : if true, the alert becomes active when updated **default=true**

Optional attributes:

- `caseTemplate` (string) : case template to use when a case is created from this alert. If the alert specifies a non-existent case template or doesn't supply one, TheHive will import the alert into a case using a case template that has the exact same name as the alert's type if it exists. For example, if you raise an alert with a type value of `splunk` and you do not provide the `caseTemplate` attribute or supply a non-existent one (for example `splink`), Base module will import the alert using the case template called `splunk` if it exists. Otherwise, the alert will be imported using an empty case (i.e. from scratch).

Attributes generated by the backend:

- `lastSyncDate` (date) : date of the last synchronization
- `case` (string) : id of the case, if created

Alert ID is computed from `type`, `source` and`sourceRef`.

Alert Manipulation

Alert methods

Get an alert

An alert's details can be retrieve using the url:

```
GET     /api/alert/:alertId
```

The alert ID is obtained by List alerts or Find alerts API.

If the parameter `similarity` is set to "1" or "true", this API returns information on cases which have similar observables. With this feature, output will contain the `similarCases` attribute which list case details with:

- artifactCount: number of observables in the original case
- iocCount: number of observables marked as IOC in original case
- similarArtifactCount: number of observables which are in alert and in case
- similarIocCount: number of IOCs which are in alert and in case

warning IOCs are observables

Examples

Get alert without similarity data:

```
curl -H 'Authorization: Bearer ***API*KEY***' http://127.0.0.1:9000/api/alert/
↪ce2c00f17132359cb3c50dfbb1901810
```

It returns:

```
{
    "_id": "ce2c00f17132359cb3c50dfbb1901810",
    "_type": "alert",
    "artifacts": [],
    "createdAt": 1495012062014,
    "createdBy": "myuser",
    "date": 1495012062016,
    "description": "N/A",
    "follow": true,
    "id": "ce2c00f17132359cb3c50dfbb1901810",
    "lastSyncDate": 1495012062016,
    "severity": 2,
    "source": "instance1",
    "sourceRef": "alert-ref",
    "status": "New",
    "title": "New Alert",
    "tlp": 2,
    "type": "external",
    "user": "myuser"
}
```

Get alert with similarity data:

```
curl -H 'Authorization: Bearer ***API*KEY***' http://127.0.0.1:9000/api/alert/
↪ce2c00f17132359cb3c50dfbb1901810?similarity=1
```

It returns:

```
{
    "_id": "ce2c00f17132359cb3c50dfbb1901810",
    "_type": "alert",
    "artifacts": [],
    "createdAt": 1495012062014,
    "createdBy": "myuser",
    "date": 1495012062016,
    "description": "N/A",
    "follow": true,
    "id": "ce2c00f17132359cb3c50dfbb1901810",
    "lastSyncDate": 1495012062016,
    "severity": 2,
    "source": "instance1",
```

(continues on next page)

(continued from previous page)

```
    "sourceRef": "alert-ref",
    "status": "New",
    "title": "New Alert",
    "tlp": 2,
    "type": "external",
    "user": "myuser",
    "similarCases": [
        {
            "_id": "AVwwrym-Rw5vhyJUfdJW",
            "artifactCount": 5,
            "endDate": null,
            "id": "AVwwrym-Rw5vhyJUfdJW",
            "iocCount": 1,
            "resolutionStatus": null,
            "severity": 1,
            "similarArtifactCount": 2,
            "similarIocCount": 1,
            "startDate": 1495465039000,
            "status": "Open",
            "tags": [
                "src:MISP"
            ],
            "caseId": 1405,
            "title": "TEST",
            "tlp": 2
        }
    ]
}
```

Create an alert

An alert can be created using the following url:

```
POST     /api/alert
```

Required case attributes (cf. models) must be provided.

If an alert with the same tuple `type`, `source` and `sourceRef` already exists, Base module will refuse to create it.

This call returns attributes of the created alert.

Examples

Creation of a simple alert:

```
curl -XPOST -H 'Authorization: Bearer ***API*KEY***' -H 'Content-Type: application/
↪json' http://127.0.0.1:9000/api/alert -d '{
  "title": "New Alert",
  "description": "N/A",
  "type": "external",
  "source": "instance1",
  "sourceRef": "alert-ref"
}'
```

It returns:

```
{
    "_id": "ce2c00f17132359cb3c50dfbb1901810",
    "_type": "alert",
    "artifacts": [],
    "createdAt": 1495012062014,
    "createdBy": "myuser",
    "date": 1495012062016,
    "description": "N/A",
    "follow": true,
    "id": "ce2c00f17132359cb3c50dfbb1901810",
    "lastSyncDate": 1495012062016,
    "severity": 2,
    "source": "instance1",
    "sourceRef": "alert-ref",
    "status": "New",
    "title": "New Alert",
    "tlp": 2,
    "type": "external",
    "user": "myuser"
}
```


Creation of another alert:


```
curl -XPOST -H 'Authorization: Bearer ***API*KEY***' -H 'Content-Type: application/
↪json' http://127.0.0.1:9000/api/alert -d '{
  "title": "Other alert",
  "description": "alert description",
  "type": "external",
  "source": "instance1",
  "sourceRef": "alert-ref",
  "severity": 3,
  "tlp": 3,
  "artifacts": [
    { "dataType": "ip", "data": "127.0.0.1", "message": "localhost" },
    { "dataType": "domain", "data": "energysoar.com", "tags": ["home", "file"] },
    { "dataType": "file", "data": "logo.svg;image/svg+xml;
↪PD94bWwgdmVyc2lvbj0iMS4wIiBlbmNvZGluZz0idXRmLTgiPz4NCjwhLS0gR2VuZXJhdG9yOiBBZG9iZSBJbGx1c3RyYXRvciA
↪", "message": "logo" }
  ],
  "caseTemplate": "external-alert"
}'
```


Merge an alert

An alert can be merge in a case using the URL:

```
POST     /api/alert/:alertId/merge/:caseId
```

Each observable of the alert will be added to the case if it doesn't exist in the case. The description of the alert will be appended to the case's description.

The HTTP response contains the updated case.

Example

Merge the alert `ce2c00f17132359cb3c50dfbb1901810` in case `AVXeF-pZmeHK_2HEYj2z`:

```
curl -XPOST -H 'Authorization: Bearer ***API*KEY***' http://127.0.0.1:9000/api/alert/
↪ce2c00f17132359cb3c50dfbb1901810/merge/AVXeF-pZmeHK_2HEYj2z
```

The call returns:

```
{
  "severity": 3,
  "createdBy": "myuser",
  "createdAt": 1488918582777,
  "caseId": 1,
  "title": "My first case",
  "startDate": 1488918582836,
  "owner": "myuser",
  "status": "Open",
  "description": "This case has been created by my custom script

  ### Merged with alert #10 my alert title

  This is my alert description",
  "user": "myuser",
  "tlp": 2,
  "flag": false,
  "id": "AVXeF-pZmeHK_2HEYj2z",
  "_id": "AVXeF-pZmeHK_2HEYj2z",
  "_type":"case"
}
```

Bulk merge alert

This API merge several alerts with one case:

```
POST     /api/alert/merge/_bulk
```

The observable of each alert listed in `alertIds` field will be imported into the case (identified by `caseId` field). The description of the case *is not* modified.

The HTTP response contains the case.

Example

Merge the alerts `ce2c00f17132359cb3c50dfbb1901810` and `a97148693200f731cfa5237ff2edf67b` in case `AVXeF-pZmeHK_2HEYj2z`:

```
curl -XPOST -H 'Authorization: Bearer ***API*KEY***' -H 'Content-Type: application/
↪json' http://127.0.0.1:9000/api/alert/merge/_bulk -d '{
  "caseId": "AVXeF-pZmeHK_2HEYj2z",
  "alertIds": ["ce2c00f17132359cb3c50dfbb1901810", "a97148693200f731cfa5237ff2edf67b"]
}'
```

The call returns:

```
{
  "severity": 3,
  "createdBy": "myuser",
  "createdAt": 1488918582777,
  "caseId": 1,
```

(continues on next page)

(continued from previous page)

```
  "title": "My first case",
  "startDate": 1488918582836,
  "owner": "myuser",
  "status": "Open",
  "description": "This case has been created by my custom script",
  "user": "myuser",
  "tlp": 2,
  "flag": false,
  "id": "AVXeF-pZmeHK_2HEYj2z",
  "_id": "AVXeF-pZmeHK_2HEYj2z",
  "_type":"case"
}
```

7.1.3 Observable

Model definition

Required attributes:

- `data` (string) : content of the observable (read only). An observable can't contain data and attachment attributes
- `attachment` (attachment) : observable file content (read-only). An observable can't contain data and attachment attributes
- `dataType` (enumeration) : type of the observable (read only)
- `message` (text) : description of the observable in the context of the case
- `startDate` (date) : date of the observable creation **default=now**
- `tlp` (number) : TLP (`0`: `white`; `1`: `green`; `2`: `amber`; `3`: `red`) **default=2**
- `ioc` (boolean) : indicates if the observable is an IOC **default=false**
- `status` (artifactStatus) : status of the observable (*Ok* or *Deleted*) **default=Ok**

Optional attributes:

- `tags` (multi-string) : observable tags

Observable manipulation

Observable methods

List Observables of a Case

Complete observable list of a case can be retrieved by performing a search:

```
POST     /api/case/artifact/_search
```

Parameters:

- `query: { "_parent": { "_type": "case", "_query": { "_id": "<<caseId>>" } } }`
- `range: all`

<<caseId>> must be replaced by case id (not the case number !)

7.1.4 Case

Model definition

Required attributes:

- `title` (text) : title of the case
- `description` (text) : description of the case
- `severity` (number) : severity of the case (1: low; 2: medium; 3: high) **default=2**
- `startDate` (date) : date and time of the begin of the case **default=now**
- `owner` (string) : user to whom the case has been assigned **default=use who create the case**
- `flag` (boolean) : flag of the case **default=false**
- `tlp` (number) : TLP (`0`: `white`; `1`: `green`; `2`: `amber`; `3`: `red`) **default=2**
- `tags` (multi-string) : case tags **default=empty**

Optional attributes:

- `resolutionStatus` (caseResolutionStatus) : resolution status of the case (*Indeterminate*, *FalsePositive*, *TruePositive*, *Other* or *Duplicated*)
- `impactStatus` (caseImpactStatus) : impact status of the case (*NoImpact*, *WithImpact* or *NotApplicable*)
- `summary` (text) : summary of the case, to be provided when closing a case
- `endDate` (date) : resolution date
- `metrics` (metrics) : list of metrics

Attributes generated by the backend:

- `status` (caseStatus) : status of the case (*Open*, *Resolved* or *Deleted*) **default=Open**
- `caseId` (number) : Id of the case (auto-generated)
- `mergeInto` (string) : ID of the case created by the merge
- `mergeFrom` (multi-string) : IDs of the cases that were merged

Case Manipulation

Case methods

Create a Case

A case can be created using the following url :

```
POST    /api/case
```

Required case attributes (cf. models) must be provided.

This call returns attributes of the created case.

Examples

Creation of a simple case:

```
curl -XPOST -H 'Authorization: Bearer ***API*KEY***' -H 'Content-Type: application/
↪json' http://127.0.0.1:9000/base/api/case -d '{
  "title": "My first case",
  "description": "This case has been created by my custom script"
}'
```

It returns:

```
{
  "severity": 3,
  "createdBy": "myuser",
  "createdAt": 1488918582777,
  "caseId": 1,
  "title": "My first case",
  "startDate": 1488918582836,
  "owner": "myuser",
  "status": "Open",
  "description": "This case has been created by my custom script",
  "user": "myuser",
  "tlp": 2,
  "flag": false,
  "id": "AVqqdpY2yQ6w1DNC8aDh",
  "_id": "AVqqdpY2yQ6w1DNC8aDh",
  "_type":"case"
}
```

Creation of another case:

```
curl -XPOST -H 'Authorization: Bearer ***API*KEY***' -H 'Content-Type: application/
↪json' http://127.0.0.1:9000/base/api/case -d '{
  "title": "My second case",
  "description": "This case has been created by my custom script, its severity is␣
↪high, tlp is red and it contains tags",
  "severity": 3,
  "tlp": 3,
  "tags": ["automatic", "creation"]
}'
```

Creating a case with Tasks & Customfields:

```
curl -XPOST -H 'Authorization: Bearer ***API*KEY***' -H 'Content-Type: application/
↪json' http://127.0.0.1:9000/base/api/case -d '{
    "title": "My first case",
    "description": "This case has been created by my custom script"
    "tasks": [{
        "title": "mytask",
        "description": "description of my task"
    }],
    "customFields": {
        "cvss": {
            "number": 9,
        },
        "businessImpact": {
            "string": "HIGH"
        }
    }
}'
```

For the `customFields` object, the attribute names should correspond to the `ExternalReference` (cvss and businessImpact in the example above) not to the name of custom fields.

7.1.5 Log

Model definition

Required attributes:

- `message` (text) : content of the Log
- `startDate` (date) : date of the log submission **default=now**
- `status` (logStatus) : status of the log (*Ok* or *Deleted*) **default=Ok**

Optional attributes:

- `attachment` (attachment) : file attached to the log

Log manipulation

Log methods

Create a log

The URL used to create a task is:

```
POST /api/case/task/<<taskId>>/log
```

<<taskId>> must be replaced by task id

Required log attributes (cf. models) must be provided.

This call returns attributes of the created log.

Examples

Creation of a simple log in task `AVqqeXc9yQ6w1DNC8aDj`:

```
curl -XPOST -H 'Authorization: Bearer ***API*KEY***' -H 'Content-Type: application/
↪json' http://127.0.0.1:9000/base/api/case/task/AVqqeXc9yQ6w1DNC8aDj/log -d '{
  "message": "Some message"
}'
```

It returns:

```
{
  "startDate": 1488919949497,
  "createdBy": "admin",
  "createdAt": 1488919949495,
  "user": "myuser",
  "message":"Some message",
  "status": "Ok",
  "id": "AVqqi3C-yQ6w1DNC8aDq",
  "_id": "AVqqi3C-yQ6w1DNC8aDq",
  "_type":"case_task_log"
}
```

If log contains an attachment, the request must be in multipart format:

```
curl -XPOST -H 'Authorization: Bearer ***API*KEY***' http://127.0.0.1:9000/base/api/
↪case/task/AVqqeXc9yQ6w1DNC8aDj/log -F '_json={"message": "Screenshot of fake site"};
↪type=application/json' -F 'attachment=@screenshot1.png;type=image/png'
```

It returns:

```
{
  "createdBy": "myuser",
  "message": "Screenshot of fake site",
  "createdAt": 1488920587391,
  "startDate": 1488920587394,
  "user": "myuser",
  "status": "Ok",
  "attachment": {
    "name": "screenshot1.png",
    "hashes": [
      "086541e99743c6752f5fd4931e256e6e8d5fc7afe47488fb9e0530c390d0ca65",
      "8b81e038ae0809488f20b5ec7dc91e488ef601e2",
      "c5883708f42a00c3ab1fba5bbb65786c"
    ],
    "size": 15296,
    "contentType": "image/png",
    "id": "086541e99743c6752f5fd4931e256e6e8d5fc7afe47488fb9e0530c390d0ca65"
  },
  "id": "AVqqlSy0yQ6w1DNC8aDx",
  "_id": "AVqqlSy0yQ6w1DNC8aDx",
  "_type": "case_task_log"
}
```

7.1.6 Task

Model definition

Required attributes:

- `title` (text) : title of the task
- `status` (taskStatus) : status of the task (*Waiting*, *InProgress*, *Completed* or *Cancel*) **default=Waiting**
- `flag` (boolean) : flag of the task **default=false**

Optional attributes:

- `owner` (string) : user who owns the task. This is automatically set to current user when status is set to *InProgress*
- `description` (text) : task details
- `startDate` (date) : date of the beginning of the task. This is automatically set when status is set to *Open*
- `endDate` (date) : date of the end of the task. This is automatically set when status is set to *Completed*

Task manipulation

Task methods

Create a task

The URL used to create a task is:

```
POST /api/case/<<caseId>>/task
```

<<caseId>> must be replaced by case id (not the case number !)

Required task attributes (cf. models) must be provided.

This call returns attributes of the created task.

Examples

Creation of a simple task in case `AVqqdpY2yQ6w1DNC8aDh`:

```
curl -XPOST -H 'Authorization: Bearer ***API*KEY***' -H 'Content-Type: application/
↪json' http://127.0.0.1:9000/base/api/case/AVqqdpY2yQ6w1DNC8aDh/task -d '{
  "title": "Do something"
}'
```

It returns:

```
{
  "createdAt": 1488918771513,
  "status": "Waiting",
  "createdBy": "myuser",
  "title": "Do something",
  "order": 0,
  "user": "myuser",
  "flag": false,
  "id":"AVqqeXc9yQ6w1DNC8aDj",
  "_id":"AVqqeXc9yQ6w1DNC8aDj",
  "_type":"case_task"
}
```

Creation of another task:

```
curl -XPOST -H 'Authorization: Bearer ***API*KEY***' -H 'Content-Type: application/
↪json' http://127.0.0.1:9000/base/api/case/AVqqdpY2yQ6w1DNC8aDh/task -d '{
  "title": "Analyze the malware",
  "description": "The malware XXX is analyzed using sandbox ...",
  "owner": "Joe",
  "status": "InProgress"
}'
```

7.1.7 Base module Model Definition

Field Types

- `string` : textual data (example “malware”).
- `text` : textual data. The difference between `string` and `text` is in the way content can be searched.`string` is searchable as-is whereas `text`, words (token) are searchable, not the whole content (example “Ten users have received this ransomware”).
- `date` : date and time using timestamps with milliseconds format.

- `boolean` : true or false
- `number` : numeric value
- `metrics` : JSON object that contains only numbers

Field can be prefixed with `multi-` in order to indicate that multiple values can be provided.

Common Attributes

All entities share the following attributes:

- `createdBy` (text) : login of the user who created the entity
- `createdAt` (date) : date and time of the creation
- `updatedBy` (text) : login of the user who last updated the entity
- `upadtedAt` (date) : date and time of the last update
- `user` (text) : same value as `createdBy` (this field is deprecated) These attributes are handled by the back-end and can't be directly updated.

7.1.8 Request formats

Base module accepts several parameter formats within a HTTP request. They can be used indifferently. Input data can be:

- a query string
- URL-encoded form
- multi-part
- JSON

Hence, the requests below are equivalent.

Query String

```
curl -XPOST 'http://127.0.0.1:9000/api/login?user=me&password=secret'
```

URL-encoded Form

```
curl -XPOST 'http://127.0.0.1:9000/api/login' -d user=me -d password=secret
```

JSON

```
curl -XPOST http://127.0.0.1:9000/api/login -H 'Content-Type: application/json' -d '{
  "user": "me",
  "password": "secret"
}'
```

Multi-part

```
curl -XPOST http://127.0.0.1:9000/api/login -F '_json=<-;type=application/json' << _
↪EOF_
{
  "user": "me",
  "password": "secret"
}
_EOF_
```

Response Format

Base module outputs JSON data.

7.1.9 User

Model definition

Required attributes:

- `login` / `id` (string) : login of the user
- `userName` (text) : Full name of the user
- `roles` (multi-userRole) : Array containing roles of the user (`read`, `write` or `admin`)
- `status` (userStatus) : `Ok` or `Locked` **default=Ok**
- `preference` (string) : JSON object containing user preference **default={}**

Optional attributes:

- `avatar` (string) : avatar of user. It is an image encoded in base 64
- `password` (string) : user password if local authentication is used

Attributes generated by the backend:

- `key` (uuid) : API key to authenticate this user (deprecated)

User Manipulation

User methods

- `with-key` (boolean)

Create a User

A user can be created using the following URL:

```
POST     /api/user
```

Required case attributes (cf. models) must be provided.

This call returns attributes of the created user.

This call is authenticated and requires admin role.

Examples

Creation of a user:

```
curl -XPOST -H 'Authorization: Bearer ***API*KEY***' -H 'Content-Type: application/
↪json' http://127.0.0.1:9000/api/user -d '{
  "login": "georges",
  "name": "Georges Abitbol",
  "roles": ["read", "write"],
  "password": "La classe"
}'
```

It returns:

```
{
  "createdBy": "myuser",
  "name":"Georges Abitbol",
  "roles": ["read", "write" ],
  "_id": "georges",
  "user": "myuser",
  "createdAt": 1496561862924,
  "status": "Ok",
  "id": "georges",
  "_type": "user",
  "has-key":false
}
```

If external authentication is used (LDAP or AD) password field must not be provided.

7.2 Automation API Guide

7.2.1 Introduction

Automation module offers a REST API that can be leveraged by various applications and programs to interact with it. The following guide describe the Automation API to allow developers to interface the powerful observable analysis engine with other SIRPs (Security Incident Response Platforms) besides Base module, TIPs (Threat Intelligence Platforms), SIEMs or scripts. Please note that the Web UI of Automation module exclusively leverage the REST API to interact with the back-end.

Note: You can use Cortex4py, the Python library we provide, to facilitate interaction with the REST API of Automation module. You need Cortex4py 2.0.0 or later as earlier versions are not compatible with Cortex 2.

All the exposed APIs share the same *request & response formats* and *authentication strategies* as described below.

There are also some transverse parameters supported by several calls, in addition to *utility APIs*.

If you want to create an analyzer, please read the How to Write and Submit an Analyzer guide.

Request & Response Formats

Automation module accepts several parameter formats within a HTTP request. They can be used indifferently. Input data can be:

- A query string
- A URL-encoded form
- A multi-part
- JSON

Hence, the requests shown below are equivalent.

Query String

```
curl -XPOST 'https://127.0.0.1/automation/api/login?user=me&password=secret'
```

URL-encoded Form

```
curl -XPOST 'https://127.0.0.1/automation/api/login' -d user=me -d password=secret
```

JSON

```
curl -XPOST https://127.0.0.1/automation/api/login -H 'Content-Type: application/json
↪' -d '{
  "user": "me",
  "password": "secret"
}'
```

Multi-part

```
curl -XPOST https://127.0.0.1/automation/api/login -F '_json=<-;type=application/json
↪' << _EOF_
{
  "user": "me",
  "password": "secret"
}
_EOF_
```

Response Format

For each request submitted, Automation module will respond back with JSON data. For example, if the authentication request is successful, Automation module should return the following output:

```
{"id":"me","name":"me","roles":["read","analyze","orgadmin"]}
```

If not, Automation module should return an authentication error:

```
{"type":"AuthenticationError","message":"Authentication failure"}
```

Authentication

Most API calls require authentication. Credentials can be provided using a **session cookie**, an **API key** or directly using HTTP **basic authentication** (if this method is specifically enabled).

Session cookies are better suited for browser authentication. Hence, **we recommend authenticating with API keys** when calling the Automation module APIs.

Generating API Keys with an orgAdmin Account

API keys can be generated using the Web UI. To do so, connect using an `orgAdmin` account then click on *Organization* and then on the `Create API Key` button in the row corresponding to the user you intend to use for API authentication. Once the API key has been created, click on `Reveal` to display the API key then click on the *copy to clipboard* button if you wish to copy the key to your system's clipboard.

If the user is not yet created, start by clicking on `Add user` to create it then follow the steps mentioned above.

Generating API Keys with a superAdmin Account

You can use a `superAdmin` account to achieve the same result as described above. Once authenticated, click on *Users* then on the `Create API Key` button in the row corresponding to the user you intend to use for API authentication. Please **make sure the user is in the right organization** by thoroughly reading its name, which is shown below the user name. Once the API key has been created, click on `Reveal` to display the API key then click on the *copy to clipboard* button if you wish to copy the key to your system's clipboard.

Authenticating with an API Key

Once you have generated an API key you can use it, for example, to list the Automation module jobs thanks to the following `curl` command:

```
### Using API key
curl -H 'Authorization: Bearer **API_KEY**' https://127.0.0.1/automation/api/job
```

As you can see in the example above, we instructed `curl` to add the *Authorization* header to the request. The value of the header is `Bearer: **API_KEY**`. So if your API key is `GPX20GUAQWwpqnhA6JpOwNGPMfWuxsX3`, the `curl` command above would look like the following:

```
### Using API key
curl -H 'Authorization: Bearer GPX20GUAQWwpqnhA6JpOwNGPMfWuxsX3' https://127.0.0.1/
↪automation/api/job
```

Using Basic Authentication

Automation module also supports basic authentication but it is disabled by default for security reasons. **If you absolutely need to use it**, you can enable it by adding `auth.method.basic=true` to the configuration file (`/etc/cortex/application.conf` by default). Once you do, restart the Automation module service. You can then, for example, list the Automation module jobs using the following `curl` command:

```
### Using basic authentication
curl -u mylogin:mypassword https://127.0.0.1/automation/api/job
```

7.2.2 Organization APIs

Automation module offers a set of APIs to create, update and list organizations.

Organization Model

An organization (org) is defined by the following attributes:

Please note that `id` and `name` are essentially the same. Also, `createdAt` and `updatedAt` are in *epoch*.

List

It is possible to list all the organizations using the following API call, which requires the API key associated with a `superAdmin` account:

```
curl -H 'Authorization: Bearer **API_KEY**' 'https://127.0.0.1/automation/api/
↪organization'
```

You can also search/filter organizations using the following query:

```
curl -H 'Authorization: Bearer **API_KEY**' -H 'Content-Type: application/json'
↪'https://127.0.0.1/automation/api/organization/_search' -d '{
  "query": {"status": "Active"}
}'
```

Both APIs supports the `range` and `sort` query parameters described in *paging and sorting details*.

Create

It is possible to create an organization using the following API call, which requires the API key associated with a `superAdmin` account:

```
curl -XPOST -H 'Authorization: Bearer **API_KEY**' -H 'Content-Type: application/json
↪' 'https://127.0.0.1/automation/api/organization' -d '{
  "name": "demo",
  "description": "Demo organization",
  "status": "Active"
}'
```

Update

You can update an organization's description and status (`Active` or `Locked`) using the following API call. This requires the API key associated with a `superAdmin` account:

```
curl -XPATCH -H 'Authorization: Bearer **API_KEY**' -H 'Content-Type: application/json
↪' 'https://127.0.0.1/automation/api/organization/ORG_ID' -d '{
  "description": "New Demo organization",
}'
```

or

```
curl -XPATCH -H 'Authorization: Bearer **API_KEY**' -H 'Content-Type: application/json
↪' 'https://127.0.0.1/automation/api/organization/ORG_ID' -d '{
  "status": "Active",
}'
```

Delete

Deleting an organization just marks it as `Locked` and doesn't remove the associated data from the DB. To "delete" an organization, you can use the API call shown below. It requires the API key associated with a `superAdmin` account.

```
curl -XDELETE -H 'Authorization: Bearer **API_KEY**' 'https://127.0.0.1/automation/
↪api/organization/ORG_ID'
```

Obtain Details

This API call returns the details of an organization as described in the *Organization model* section.

```
curl -H 'Authorization: Bearer **API_KEY**' 'https://127.0.0.1/automation/api/
↪organization/ORG_ID'
```

Let's assume that the organization we are seeking to obtain details about is called *demo*. The `curl` command would be:

```
curl -H 'Authorization: Bearer **API_KEY**' 'https://127.0.0.1/automation/api/
↪organization/demo'
```

and it should return:

```
{
  "id": "demo",
  "name": "demo",
  "status": "Active",
  "description": "Demo organization",
  "createdAt": 1520258040437,
  "createdBy": "superadmin",
  "updatedBy": "superadmin",
  "updatedAt": 1522077420693
}
```

List Users

As mentioned above, you can use the API to return the list of **all** the users declared withing an organization. For that purpose, use the API call shown below with the API key of an `orgAdmin` or `superAdmin` account. It supports the `range` and `sort` query parameters declared in *paging and sorting details*.

```
curl -H 'Authorization: Bearer **API_KEY**' 'https://127.0.0.1/automation/api/
↪organization/ORG_ID/user'
```

and should return a list of *users*.

If one wants to filter/search for some users (active ones for example), there is a search API to use as below:

```
curl -XPOST -H 'Authorization: Bearer **API_KEY**' -H 'Content-Type: application/json
↪' 'https://127.0.0.1/automation/api/organization/ORG_ID/user/_search' -d '{
  "query": {}
}'
```

It also supports the `range` and `sort` query parameters declared in *paging and sorting details*.

List Enabled Analyzers

To list the analyzers that have been enabled within an organization, use the following API call with the API key of an `orgAdmin` user:

```
curl -H 'Authorization: Bearer **API_KEY**' 'https://127.0.0.1/automation/api/analyzer
↪'
```

It should return a list of *Analyzers*.

Please note that this API call does not display analyzers that are disabled. It supports the `range` and `sort` query parameters declared in *paging and sorting details*.

7.2.3 User APIs

The following section describes the APIs that allow creating, updating and listing users within an organization.

User Model

A user is defined by the following attributes:

List All

This API call allows a `superAdmin` to list and search all the users of all defined organizations:

```
curl -H 'Authorization: Bearer **API_KEY**' 'https://127.0.0.1/automation/api/user'
```

This call supports the `range` and `sort` query parameters declared in *paging and sorting details*.

List Users within an Organization

This call is described in *organization APIs*.

Search

This API call allows a `superAdmin` to perform search on the user accounts created in a Automation module instance:

```
curl -XPOST -H 'Authorization: Bearer **API_KEY**' -H 'Content-Type: application/json
↪' 'https://127.0.0.1/automation/api/user/_search' -d '{
  "query": {}
}'
```

This call supports the `range` and `sort` query parameters declared in *paging and sorting details*

Create

This API calls allows you to programmatically create user creation. If the call is made by a `superAdmin` user, the request must specify the organization to which the user belong in the `organization` field.

If the call is made by an `orgAdmin` user, the value of `organization` field must be the same as the user who makes the call: `orgAdmin` users are allowed to create users only in their organization.

```
curl -XPOST -H 'Authorization: Bearer **API_KEY**' -H 'Content-Type: application/json
↪' 'https://127.0.0.1/automation/api/user' -d '{
  "name": "Demo org Admin",
  "roles": [
    "read",
    "analyze",
    "orgadmin"
  ],
  "organization": "demo",
  "login": "demo"
}'
```

If successful, the call returns a JSON object representing the created user as described *above*.

```
{
  "id": "demo",
  "organization": "demo",
  "name": "Demo org Admin",
  "roles": [
    "read",
    "analyze",
    "orgadmin"
  ],
  "status": "Ok",
  "createdAt": 1526050123286,
  "createdBy": "superadmin",
  "hasKey": false,
  "hasPassword": false
}
```

Update

This API call allows updating the writable attributed of a user account. It's available to users with `superAdmin` or `orgAdmin` roles. Any user can also use it to update their own information (but obviously not their roles).

```
curl -XPATCH -H 'Authorization: Bearer **API_KEY**' -H 'Content-Type: application/json
↪' 'https://127.0.0.1/automation/api/user/USER_LOGIN' -d '{
  "name": "John Doe",
  "roles": [
    "read",
    "analyze"
  ],
  "status": "Locked"
}'
```

It returns a JSON object representing the updated user as described *above*.

Get Details

This call returns the user details. It's available to users with `superAdmin` roles and to users in the same organization. Every user can also use it to read their own details.

```
curl -H 'Authorization: Bearer **API_KEY**' 'https://127.0.0.1/automation/api/user/
↪USER_LOGIN'
```

It returns a JSON object representing the user as described *previously*.

Set a Password

This call sets the user's password. It's available to users with `superAdmin` or `orgAdmin` roles. Please note that the request needs to be made using HTTPS with a valid certificate on the server's end to prevent credential sniffing or other PITM (Person-In-The-Middle) attacks.

```
curl -XPOST -H 'Authorization: Bearer **API_KEY**' -H 'Content-Type: application/json
↪' 'https://127.0.0.1/automation/api/user/USER_LOGIN/password/set' -d '{
  "password": "SOMEPASSWORD"
}'
```

If successful, the call returns 204 (success / no content).

Change a password

This call allows a given user to change only **their own** existing password. It is available to all users including `superAdmin` and `orgAdmin` ones. Please note that if a `superAdmin` or an `orgAdmin` needs to update the password of another user, they must use the `/password/set` call described in the previous subsection.

```
curl -XPOST -H 'Authorization: Bearer **API_KEY**' -H 'Content-Type: application/json
↪' 'https://127.0.0.1/automation/api/user/USER_LOGIN/password/change' -d '{
  "currentPassword": "password",
  "password": "new-password"
}'
```

If successful, the call returns 204 (success / no content).

Set and Renew an API Key

This calls allows setting and renewing the API key of a user. It's available to users with `superAdmin` or `orgAdmin` roles. Any user can also use it to renew their own API key. Again, the request needs to be made using HTTPS with a valid certificate on the server's end to prevent credential sniffing or other PITM (Person-In-The-Middle) attacks. You know the drill ;-)

```
curl -XPOST -H 'Authorization: Bearer **API_KEY**' 'https://127.0.0.1/automation/api/
↪user/USER_LOGIN/key/renew'
```

If successful, it returns the generated API key in a `text/plain`response.

Get an API Key

This calls allows getting a user's API key. It's available to users with `superAdmin` or `orgAdmin` roles. Any user can also use it to obtain their own API key.

```
curl -H 'Authorization: Bearer **API_KEY**' 'https://127.0.0.1/automation/api/user/
↪USER_LOGIN/key'
```

If successful, the generated API key is returned in `text/plain`response

Revoke an API Key

This calls allow revoking a user's API key. This calls allow revoking a user's API key.

```
curl -XDELETE -H 'Authorization: Bearer **API_KEY**' 'https://127.0.0.1/automation/
↪api/user/USER_LOGIN/key'
```

A successful request returns nothing (HTTP 200 OK).

7.2.4 Job APIs

The following section describes the APIs that allow manipulating jobs. Jobs are basically submissions made to analyzers and the resulting reports.

Job Model

A job is defined by the following attributes:

List and Search

This call allows a user with `read`,`analyze` or `orgAdmin` role to list and search all the analysis jobs made by their organization.

If you want to list all the jobs:

```
curl -XPOST -H 'Authorization: Bearer **API_KEY**' 'https://127.0.0.1/automation/api/
↪job/_search?range=all'
```

If you want to list 10 jobs:

```
curl -XPOST -H 'Authorization: Bearer **API_KEY**' 'https://127.0.0.1/automation/api/
↪job/_search'
```

If you want to list 100 jobs:

```
curl -XPOST -H 'Authorization: Bearer **API_KEY**' 'https://127.0.0.1/automation/api/
↪job/_search?range=0-100'
```

If you want to search jobs according to various criteria:

```
curl -XPOST -H 'Authorization: Bearer **API_KEY**' -H 'Content-Type: application/json
↪' 'https://127.0.0.1/automation/api/job/_search' -d '{
  "query": {
    "_and": [
      {"status": "Success"},
      {"dataType": "ip"}
    ]
```

(continues on next page)

(continued from previous page)

```
  }
}'
```

This call supports the `range` and `sort` query parameters declared in *paging and sorting details*

Get Details

This call allows a user with `read`,`analyze` or `orgAdmin` role to get the details of a job. It does not fetch the job report.

```
curl -H 'Authorization: Bearer **API_KEY**' 'https://127.0.0.1/automation/api/job/JOB_
↪ID'
```

It returns a JSON response with the following structure:

```
{
  "id": "AWNei4vH3rJ8unegCPB9",
  "analyzerDefinitionId": "Abuse_Finder_2_0",
  "analyzerId": "220483fde9608c580fb6a2508ff3d2d3",
  "analyzerName": "Abuse_Finder_2_0",
  "status": "Success",
  "data": "8.8.8.8",
  "parameters": "{}",
  "tlp": 0,
  "message": "",
  "dataType": "ip",
  "organization": "demo",
  "startDate": 1526299593923,
  "endDate": 1526299597064,
  "date": 1526299593633,
  "createdAt": 1526299593633,
  "createdBy": "demo",
  "updatedAt": 1526299597066,
  "updatedBy": "demo"
}
```

Get Details and Report

This call allows a user with `read`,`analyze` or `orgAdmin` role to get the details of a job including its report.

```
curl -H 'Authorization: Bearer **API_KEY**' 'https://127.0.0.1/automation/api/job/JOB_
↪ID/report'
```

It returns a JSON response with the structure below. If the job is not yet completed, the `report` field contains a string representing the job status:

```
{
  "id": "AWNei4vH3rJ8unegCPB9",
  "analyzerDefinitionId": "Abuse_Finder_2_0",
  "analyzerId": "220483fde9608c580fb6a2508ff3d2d3",
  "analyzerName": "Abuse_Finder_2_0",
  "status": "Success",
  "data": "8.8.8.8",
  "parameters": "{}",
```

(continues on next page)

(continued from previous page)


```
  "tlp": 0,
  "message": "",
  "dataType": "ip",
  "organization": "demo",
  "startDate": 1526299593923,
  "endDate": 1526299597064,
  "date": 1526299593633,
  "createdAt": 1526299593633,
  "createdBy": "demo",
  "updatedAt": 1526299597066,
  "updatedBy": "demo",
  "report": {
    "summary": {
      "taxonomies": [
        {
          "predicate": "Address",
          "namespace": "Abuse_Finder",
          "value": "network-abuse@google.com",
          "level": "info"
        }
      ]
    },
    "full": {
      "abuse_finder": {
        "raw": "...",
        "abuse": [
          "network-abuse@google.com"
        ],
        "names": [
          "Google LLC",
          "Level 3 Parent, LLC"
        ],
        "value": "8.8.8.8"
      }
    },
    "success": true,
    "artifacts": []
  }
}
```


Wait and Get Job Report

This call is similar the one described above but allows the user to provide a timeout to wait for the report in case it is not available at the time the query was made:

```
curl -H 'Authorization: Bearer **API_KEY**' 'https://127.0.0.1/automation/api/job/JOB_
↪ID/waitreport?atMost=1minute'
```

The `atMost` is a duration using the format `Xhour`, `Xminute` or `Xsecond`.

Get Artifacts

This call allows a user with `read`,`analyze` or `orgAdmin` role to get the extracted artifacts from a job if such extraction has been enabled in the corresponding analyzer configuration. Please note that extraction is imperfect and you might have inconsistent or incorrect data.

```
curl -H 'Authorization: Bearer **API_KEY**' 'https://127.0.0.1/automation/api/job/JOB_
↪ID/artifacts'
```

It returns a JSON array with the following structure:

```
[
  {
    "dataType": "ip",
    "createdBy": "demo",
    "data": "8.8.8.8",
    "tlp": 0,
    "createdAt": 1525432900553,
    "id": "AWMq4tvLjidKq_asiwcl"
  }
]
```

Delete

This API allows a user with `analyze` or `orgAdmin` role to delete a job:

```
curl -XDELETE -H 'Authorization: Bearer **API_KEY**' 'https://127.0.0.1/automation/
↪api/job/JOB_ID'
```

This marks the job as `Deleted`. However the job's data is not removed from the database.

7.2.5 Analyzer APIs

The following section describes the APIs that allow manipulating analyzers.

Analyzer Model

An analyzer is defined by the following attributes:

Enable

This call allows a user with an `orgAdmin` role to enable an analyzer.

```
curl -XPOST -H 'Authorization: Bearer **API_KEY**' 'https://127.0.0.1/automation/api/
↪organization/analyzer/:analyzerId' -d '{
  "name": "Censys_1_0",
  "configuration": {
    "uid": "XXXX",
    "key": "XXXXXXXXXXXXXXXXXXXX",
    "proxy_http": "http://proxy:9999",
    "proxy_https": "http://proxy:9999",
    "auto_extract_artifacts": false,
    "check_tlp": true,
    "max_tlp": 2
  },
  "rate": 1000,
  "rateUnit": "Day",
  "jobCache": 5
}'
```

List and Search

These calls allow a user with a `analyze` or `orgAdmin` role to list and search all the enabled analyzers within the organization.

```
curl -H 'Authorization: Bearer **API_KEY**' 'https://127.0.0.1/automation/api/analyzer
↪'
```

or

```
curl -XPOST -H 'Authorization: Bearer **API_KEY**' -H 'Content-Type: application/json
↪' 'https://127.0.0.1/automation/api/analyzer/_search' -d '{
  "query": {}
}'
```

Both calls supports the `range` and `sort` query parameters declared in *paging and sorting details*, and both return a JSON array of analyzer objects as described in *Analyzer Model section*.

If called by a user with only an `nalyzer` role, the `configuration` attribute is not included on the JSON objects.

Get Details

This call allows a user with a `analyze` or `orgAdmin` role to get an analyzer's details.

```
curl -H 'Authorization: Bearer **API_KEY**' 'https://127.0.0.1/automation/api/
↪analyzer/ANALYZER_ID'
```

It returns a analyzer JSON object as described in *Analyzer Model section*.

If called by a user with only an `nalyzer` role, the `configuration` attribute is not included on the JSON objects.

Get By Type

This call is mostly used by TheHive and allows to quickly get the list of analyzers that can run on the given datatype. It requires an `analyze` or `orgAdmin` role.

```
curl -H 'Authorization: Bearer **API_KEY**' 'https://127.0.0.1/automation/api/
↪analyzer/type/DATA_TYPE'
```

It returns a JSON array of analyzer objects as described in *Analyzer Model section* without the `configuration` attribute, which could contain sensitive data.

Update

This call allows an `orgAdmin` user to update the `name`, `configuration` and `jobCache` of an enabled analyzer.

```
curl -XPATCH -H 'Authorization: Bearer **API_KEY**' -H 'Content-Type: application/json
↪' 'https://127.0.0.1/automation/api/analyzer/ANALYZER_ID' -d '{
  "configuration": {
    "key": "XXXXXXXXXXXXXXXXXXXXXXXXXXXXX",
    "polling_interval": 60,
    "proxy_http": "http://localhost:8080",
    "proxy_https": "http://localhost:8080",
    "auto_extract_artifacts": true,
    "check_tlp": true,
```

(continues on next page)

(continued from previous page)

```
    "max_tlp": 1
  },
  "name": "Shodan_Host_1_0",
  "rate": 1000,
  "rateUnit": "Day",
  "jobCache": null
}'
```

It returns a JSON object describing the analyzer as defined in *Analyzer Model section*.

Run

This API allows a user with a `analyze` or `orgAdmin` role to run analyzers on observables of different datatypes.

For `file` observables, the API call must be made as described below:

```
curl -XPOST -H 'Authorization: Bearer **API_KEY**' -H 'Content-Type: application/json
↪' 'https://127.0.0.1/automation/api/analyzer/ANALYZER_ID/run' \
  -F 'attachment=@/path/to/observable-file' \
  -F '_json=<-;type=application/json' << _EOF_
  {
    "dataType":"file",
    "tlp":0
  }
_EOF_
```

for all the other types of observerables, the request is:

```
curl -XPOST -H 'Authorization: Bearer **API_KEY**' -H 'Content-Type: application/json
↪' 'https://127.0.0.1/automation/api/analyzer/ANALYZER_ID/run' -d '{
  "data":"8.8.8.8",
  "dataType":"ip",
  "tlp":0,
  "message": "A message that can be accessed from the analyzer",
  "parameters": {
    "key1": "value1",
    "key2": "value2"
  }
}'
```

This call will fetch a similar job from the cache, and if it finds one, it returns it from the cache, based on the duration defined in `jobCache` attribute of the analyzer.

To force bypassing the cache, one can add the following query parameter: `force=1`

```
curl -XPOST -H 'Authorization: Bearer **API_KEY**' -H 'Content-Type: application/json
↪' 'https://127.0.0.1/automation/api/analyzer/ANALYZER_ID/run?force=1' -d '{
  "data":"8.8.8.8",
  "dataType":"ip",
  "tlp":0,
  "message": "A message that can be accessed from the analyzer",
  "parameters": {
    "key1": "value1",
    "key2": "value2"
  }
}'
```

Disable

This API allows an `orgAdmin` to disable an existing analyzer in their organization and delete the corresponding configuration.

```
curl -XDELETE -H 'Authorization: Bearer **API_KEY**' 'https://127.0.0.1/automation/
↪api/analyzer/ANALYZER_ID'
```

7.2.6 Miscellaneous APIs

Paging and Sorting

All the `search` API calls allow sorting and paging parameters, in addition to a query in the request's body. These calls usually have URLs ending with the `_search` keyword but that's not always the case.

The followings are query parameters:

- `range`: `all` or `x-y` where `x` and `y` are numbers (ex: 0-10).
- `sort`: you can provide multiple sort criteria such as: `-createdAt` or `+status`.

Example:

```
curl -XPOST -H 'Authorization: Bearer **API_KEY**' -H 'Content-Type: application/json
↪' 'http://127.0.0.1/automation/api/organization/ORG_ID/user?range=0-10&sort=-
↪createdAt&sort=+status' -d '{
  "query": {}
}'
```