TaranisNG

Project and technical specification

High level overview of deliverables and design goals

This chapter is just an overview and does not cover 100% of the required functionality. For detailed technical specification, please see the following chapters.

Aim is to develop "Taranis NG" - a tool for collecting news items, learning about vulnerabilities and other security related news, and publishing advisories.

The software will be free and open-source, designed with security in mind, and will allow collaboration between CSIRT teams. Moreover, it's dependencies will also be open-source, such as there must not be a hard requirement for a commercial SQL database, libraries, or OS. Preferred OS platform of choice is deb package based Linux distribution.

The software will be used by operators to perform following tasks:

- **Collect the news items from various sources** (transport: http/https, pop/imap, file storage; format: html page, RSS feed, e-mail body, e-mail attachment - pdf, word, various file formats),
- **Group** the news items according to pre-defined rules before presenting them to the operator (operator has the option to group and un-group items manually as well)
- **Analyze** highlighted news items / item groups in process of creating an advisory
    - Automatically enrich the data on demand (download details from NVD, whois?, information from partners using other instances of the same software)
    - Easily search in the previously published advisories so that we don't report the same vulnerability twice,
- **Create** text of an advisory with multiple fields and attributes (report item)
- **Distribute** the advisories in form of products (website text, PDF, e-mail, in-app text, MISP)
    - To particular recipients according to CPE code field of the advisory, as seen in the CPE dictionary
    - Immediate warnings
    - Periodic advisories
- **Reports**, statistics, wallboards
    - supporting management
    - supporting operators
- Configuration interface

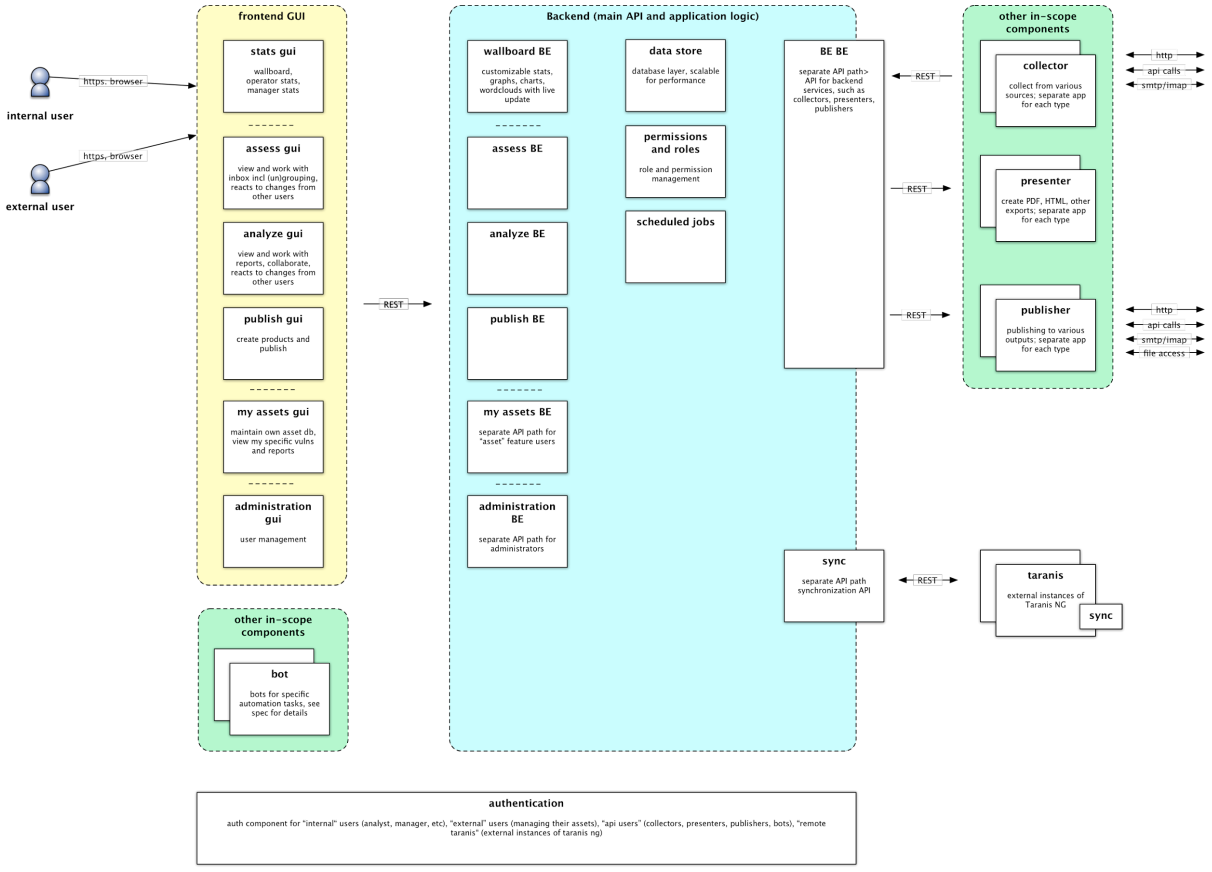
The software will also be used by external users to perform following tasks:

- register their assets and asset groups
- receive notifications about vulnerabilities related to their assets

- search and browse through the advisories archive according to the product group they have permission for.

The software will be split to following components:

- frontend (GUI) tending to multiple types of users
- backend (REST API)
- external and/or local authentication
- collectors (will use the same REST API to connect to the rest of the system)
- presenters (ability to create plain text, template PDF or other output)
- publishers (send the resulting file via mail, upload to directory, upload to CMS, twitter, MISP)
- bots (automate some subset of tasks via API; actions performed by bots are the same as the actions performed by regular users through GUI); bots receive their configuration from the backend so there should be a part of the backend API for storing, editing and retrieving their configuration, and possibly a part of the GUI which will expose the settings to administrators.



Roles: various parts of the API and GUI will only be available to users that have certain roles and permissions.

Collaboration: the software will be used by multiple users within the same organization, but at the same time, it will allow collaboration of multiple teams, partners, each running their

own instance of the software. For each partner, allowed sharing types may be enabled/disabled:

- sharing "+1/-1" for news items which will allow the operator to see interesting news items in real time
- sharing comments to the news items themselves
- sharing news items themselves
- sharing particular analysis items and their contents / attributes
- sharing the grouping rules, sources, and other interesting parts of the static and runtime configuration

Other deliverables and conditions

The software delivery will also include

- English and Slovak localization
- a user manual in English and Slovak, in editable format such as Markdown format. The manual could be integrated in product, but must be article based and cover all common/typical usage scenarios - popup help is not enough.
- an installation role for Ansible and a short installation manual,
- validation rules (regex) for all API call parameters.
- unit tests for the API.
- upgrade script from existing Taranis from NCSC-NL, that will convert the basic configuration (configured feeds, groups, users). Data migration (old analysis) is not needed.

The supplier will draft the UI/UX for the Frontend GUI module and subsequently tailor it according to feedback of the customer, subject to customer approval before proceeding with the development.

All required security requirements will be met.

Project will be developed and delivered gradually in increments, based on a set of predefined milestones. At each milestone, conformance to the requirements will be evaluated, including functionality, source code, security and any other quality indicators defined in the project documentation. The source code will best be delivered by providing an access to the source control management system, such as GIT repository. Customer will perform their own functional and security testing. Delivery will only be accepted when any identified issues are resolved and no new issues will arise from the following mandatory retest.

Customer will organize at most two rounds of testing for each milestone. For every subsequent testing round needed, the supplier will offer a predefined discount.

Security

- Design practices and considerations

    - separate components: GUI will be clearly separated from the API backend, bots and GUI users use the same API
    - frameworks: both frontend and backend will make use of existing frameworks, that minimize the opportunity for human error and security issues.
    - backend framework will provide ORM, permission system, and escaping for I/O, especially exports.
    - frontend framework will provide escaping for output
- All HTTP endpoints will use TLS. GUI frontend will use free certificates such as LetsEncrypt or uploaded certificates. Clients will validate certificates. Technical connections will have option to use preloaded self-signed certificates.
- Software development practices and considerations
    - all inputs will be validated **as soon as possible**. Validation does NOT include escaping for SQL, etc.
    - all data will be escaped **as LATE as possible** (just before output to HTML, PDF, txt, SQL, XML, CSV, shell execution, …), data always escaped even if the data source is thought to be free of special characters at the time of development. Escaping should be done by framework if possible.
    - special care will be taken to avoid race conditions and deadlocks. Any valid or invalid API calls may occur concurrently, so the API call implementation will use locking as necessary. This covers database access, as well as filesystem usage, resource allocation, etc.
    - received file uploads, if at all necessary, will use randomly generated safe filenames. Original file and path names, if needed, may only be stored in a database together with other upload metadata.
- Permissions, ACL entries: the API part will provide services according to permissions and ACL entries. Both of these may be assigned to a role, user, or everyone. Permissions implement MAC (mandatory access control), while ACL entries implement DAC (discretionary access control). Existence of permission to perform each and every particular action will be checked server-side. ACLs are also checked server side. Most of these tests will be performed directly by the framework. Data will be further filtered. More detailed definition of roles and permissions is given in the Permissions, ACLs, and access control chapter.
- Furthermore, the API will be segmented so that the most distinct use cases can be treated differently at WAF / reverse proxy / load balancer level. Following distinct API path groups are defined:
    - wallboard+analyst API (for regular tasks of CSIRT, and for bots)
    - administrator API (configuration and management)
    - assets API (for external users who maintain their list of assets)
    - backend API (for backend services)
    - connector API (for taranis to taranis links)
- All actions of all users will be logged in a *searchable*, *exportable* and human readable event log. The event log contains these types of events:
    - user action (logged for every action initiated by user calling an API call)
    - security warning, security error (logged for every unauthorized access, and other security related events)
    - system information, system warning, system error

- Audit: the software will pass source code review, and penetration testing, and all identified vulnerabilities will be resolved prior accepting the software. More about testing can be found in “Other deliverables and conditions”.

Spec: Frontend GUI and user scenarios

Primary GUI will be a responsive web interface, equally usable through desktop and tablet browsers, and it should not be fixed for any particular resolution or pixel density (it may be used from tablet screens up to 8k TV). Also, the browser zooming feature must work, so that people with different eyesight can successfully use the software. The GUI should work on the most popular browsers, such as reasonably recent versions of Google Chrome, Microsoft Edge, Mozilla Firefox, Safari, etc.

It will tend to the following user groups / roles:

- internal users
    - security analysts (the most frequent users)
    - system administrators (multiple roles for each separate part of system management, such as: configuration of feeds, internal user/role/permission management, external organization/user management, TLS certificate management, ...)
    - managers/supervisors
    - wallboard / team user (multiple roles with access to on-screen wallboards in SOC, inbox, and other relevant information)
- external organization users
    - users (they manage their assets, view personalized advisories, view general advisories)
    - external org. administrators (they manage users for their own organization)

The interface will allow **keyboard shortcuts** and **gestures** for the most common **actions**, including movement through the inbox, dismissing or upvoting entries, opening a new analysis. The shortcuts will not be hardcoded, but there will be a default, and a per-user, section in settings, where the user can bind these actions to shortcuts.

The interface will support **i18n** and will be easily translatable to various languages. The internationalization will use format strings with named parameters, or editable templates, so that order of displayed values could be customized for different languages. The interface language preference will be persisted on per-user basis. The first version does not need to support RTL languages. The i18n will be done using GNU gettext .pot/.po files.

The screen layout will be laid out by a **template engine** and will contain these elements:

- header/footer (included from separate templates to allow easy customization)
- menu to select modules
- module contents
- currently logged user and role, ability to logout

The software will allow simultaneous changes to single vulnerability (adding comments, etc.) from various internal users. GUI will react to changes made by other operators, and will try to resolve all conflicts automatically on background. If not possible, user must be notified and be allowed to resolve conflict without losing information. Data from remote taranis instances

is NOT synchronized with internal products but displayed alongside, in the same view, with possibility for the operator to re-use the block. Thus, bi-directional synchronization with remote sides is not necessary.

The skin and overall appearance can be customized by end user - globally per site (preferably by allowing changes to templates, less preferably by predefined configurables such as logo & colors).

Most relevant form elements will use a html5 spellcheck attribute to enable spell checking. The preference of spell checking will be saved on per-user basis, and will be on by default.

The frontend will use URI to highlight the currently visited section and item, so that the entries of any section are shareable via a (perma-)link at least for users of the same TaranisNG instance. This will also allow for “open in new tab” for TaranisNG sections, and action dialogs/views.

The GUI will only display those components and controls, for which the user has permission. This is not a replacement for the proper security checking on the backend, but a convinience feature for users.

Where appropriate and possible, the GUI will perform first level of input validation with a visual cue to the user.

Wallboards and statistics module

This module calculates and displays the statistics intended for the following different audiences:

- operators (wallboard)
- supervisors (performance of operators, fulfillment of KPI)
- external users (statistics related to their assets and situation)
- public (general vulnerability statistics)

The wallboard values will be automatically updated on the backend. They will be calculated on background, rather than at the time of API request, for efficiency reasons. The frontend will display the recent information in near real time - one update per minute is sufficient. Full page refresh is not acceptable.

The operator’s wallboard includes:

- keyword cloud
- news items
    - number of received news items (time based graph - hours and per day)
    - number of unread news items (current value and time based graph - hours and per day)
    - number of currently open news items (current value and time based graph - hours and per day)

    - average and maximum time to process (max will be per day; current value and time based graph)
- statistics related to report items
    - (numbers by phase, time based graphs - hours and per day)
- product statistics
- a limited number of other wallboard items may be specified later

The supervisors statistics include:

- overall and per-person statistics for basic activities (numeric and time based graph)
- KPI indicators (to be specified in more detail), for example
    - user's efficiency/amount of work per user and type {read news items, dismissed news items, used news items, created report items, contributed report items, published products, etc.} (in time)
- number of users, etc.
- a limited number of other wallboard items may be specified

The unrestricted and external user's statistics include:

- keyword cloud
- a limited number of other wallboard items may be specified

The statistics also include:

- preparation of weekly and monthly reports (security status view, manager/team supervisor view etc.)
- security status dashboard
- a limited number of other wallboard items may be specified

The module will also provide some unrelated communication helpers:

- a global chat for logged-in users (available from any screen)
- a global "server message" pop-up for information such as maintenance notice

"Assess" module

The module will display an inbox - a list of **news items** obtained from various sources intermixed with **news item aggregates** for those items that have been joined together (either automatically or by user). The list will be searchable and sortable by any data, associated with entry. The list entry will include at least these items:

- title of the article or aggregate
- in case of aggregate, indicator of aggregated entries (will unfold to the respective entries on click)
- preview of the text of the article and its details (language, timestamp, author, ...)
- controls (to be specified later; especially upvote/downvote, mark as read/unread, delete, important, create analysis, send by e-mail)
- checkbox for group operations and group controls (fold/unfold, upvote/downvote, mark as read/unread, delete, important, create analysis, send by e-mail)

The module will include filter, search and sorting features, which will allow at least:

- chose items from particular source group (tab based)
- search by string in title, full text, comments of the news item / aggregate
- filter by state (unread, read/deleted, important, in analysis)
- order by date or community score
- upvote / downvote a news item / news item aggregate

Pagination or load on demand will be used if the list of results is too large.

Module will allow reading the stored version of the article, as well as jumping to the original source.

The Assess module will allow for a quick search through all completed report items (for instance in the subject, text, CVE number, etc.).

The Assess module will automatically apply selected wordlists and highlight the identified keywords on screen. Which wordlists are enabled for such comparison is a configurable feature, where every user can choose from their available word lists and enable/disable them. The same UI will allow for “re-match” in case the word list has changed.

There will be an option to match word-list against a single / several assess-items anytime on demand.

See also: Spec: collectors.

“Analyze” module

The module will display an inbox - a list of all currently open **report items** (most usually vulnerabilities). The list will be searchable (also by keywords in various fields including IDs, CVEs, etc; also by time intervals for time of creation, and time of last edit) and sortable (also by created_on asc/desc, content_changed asc/desc). It will support paging if many active report items are present, but must display large number of entries on single page by default to prevent unnecessary paging. As mentioned in the general overview, keyboard shortcuts will be available to navigate the list.

New report item may be created directly in the “Analyze” module, or by selecting one or more news items or news aggregates within “Assess” module and selecting the appropriate action to create analysis. In this case, some entries will be automatically pre-parsed from the news items if possible / appropriate.

A single report item will display a title, a state and basic controls, and a set of information blocks with various kinds of information.

One information block displays and optionally allows editing of one report item attribute. Some examples of report item information blocks are: report item rich-text description, CVSS score, affected systems matrix, linked news items and news item aggregates (internal link), linked report items (internal link), URLs (external link), file attachments (with ability to

upload/download), etc. The information block will be subtly labeled with date/time, and its author(s) along the edge of the block, and visually distinct depending on the source (internal user / external taranis through connector).

The information blocks will be grouped into an information block group. The block groups will have a subtle label and will contain internally and externally originated information of that type. Any block group can be collapsed by the operator to conserve screen space. There will be controls allowing to collapse all / expand all. The ordering of the information blocks and their labels are defined in a report item type (type to be selected when creating the report item).

The information blocks inside the block group will be ordered. Two ordering modes will be available and switchable from GUI: time, then author, and author, then time. Ordering by author will always display current user first, then other local users in alphabetical order, then remote Taranis instances in alphabetical order. The ordering preference will be persisted on per-user basis.

New information blocks will appear in near real time as they are created by internal users or external Taranis instances. Changes will be synchronized to all users and external instances. There will be a visual indication when the content of information block changes, both at the block and at the block group. If possible, the system will allow co-editing of a single item (this only applies to concurrent local users; as mentioned before, remote content is always displayed separately from locally originated one). Otherwise, a visual indication of currently edited item will be displayed as soon as the entry is being edited by someone else.

Information blocks from external Taranis instances are read-only. External connector will not disclose login names of users who created the items on remote side. Name of the remote Taranis instance will be displayed instead.

New information blocks (i.e. report item attributes) may be added to the block group through the GUI, subject to cardinality limits defined in the report item type. Existing blocks originating from the internal users may be edited or deleted within the GUI, with the exception of a few mandatory fields such as title. As mentioned in the general overview, keyboard shortcuts will be available to navigate and edit the data.

New information blocks may also be added in bulk, either by pasting the text into a large text field, or by using file upload, with each entry on a separate line. This feature will be purely frontend based, with frontend calling the appropriate number of API calls as if the entries were inserted manually one by one. The same cardinality rules apply.

The following block types are to be recognized:

- internal comments, description, recommendations, resolution, etc. (rich text editor such as tinymce, multiple instances of this field are allowed; templates must be available)
- counters with up/down button (could be used for versioning)

- related report items, related news items (“type to search” searchable dropdown to add new related item, list of item links added, possibility to delete a link)
- custom taxonomy (there will be an editable list of taxonomies elsewhere in the administration GUI; here a searchable dropdown will be displayed for selected taxonomy; these can be used for confidentiality, TLP, sector, threat level, and much more)
- date, datetime, time inputs
- CVSS(v2, v3) score and CVSS(v3) calculator (will allow entry by number, and entry by vector; will still display the number if vector is unavailable; will always update the number if calculator entries are modified)
- CVE number (multiple CVE numbers allowed)
- affected systems (searchable CPE editor, multiple CPE codes allowed)
- file attachment (multiple allowed; all security considerations for file storage apply)
- URLs and hostnames
- IP addresses (IPv4, IPv6)
- email addresses

Some other block types will be added later, such as “Taranis Matrix” calculator. For this reason both the backend, and frontend implementation of blocks must be as open and simple to change as possible. Se e also: News item and report item attributes.

The rich text editor must provide the following advanced features:

- ability to paste and strip the invisible formating
- ability to sanitize URLs and IP addresses (10.1.2.4 → 10[.]1[.]2[.]3, https://a.b.c.d → hxxps://a[.]b[.]c[.]d)
- marking / highlighting selected parts of the text. This feature will be used for marking special parts of the text for the presenter.

The “affected systems” block will display a count of matched assets from the “my assets” component, so that analyst can directly see the impact of vulnerability on current constituency.

Title of the report item should allow chosing a prefix from dropdown (for example [URGENT]).

Report items are versioned, and there is a possibility to view the edit history and display side-by-side changes.

There will be a tab for “linked items” which will display all linked items regardless of type (e.g. news items, report items, products, external links), clustered by id-type. There will be a feature to add new linked item, either as a text, or for internal contents by search-pulldown. Shortname for IDs: Taranis-IDs and foreign IDs should be displayable like href: displayed text vs clickable Link (to foreign system) (e.g. text: "LZ#201812345", link: "https://myOTRS.de/ticket/show?ticketid=201812345")

- custom ID-templates should be creatable in Taranis-GUI

- on hovering over any IDs a tooltip will be displayed describing the meaning of that ID (e.g. CVE: short summary of that vulnerability)
- tooltips are copied into clipboard on click on tooltip / on click on "copy-button" inside that tooltip user "jumps" to edit-/configure-view of that tooltip on click on "edit-button" inside that tooltip (depending on user rights)

The “analyze” module will contain a Delegate tab to allow delegation of the analysis via e-mail. The possible delegations from the configuration will be displayed in the form of check boxes with the title of delegation. The wordlist matches will pre-populate the check lists. There will be a button labeled “Delegate” and after using it, the time of delegation is displayed near the used check boxes. The body of delegation must contain details of the currently open report item.

Finalized report items are marked by the analyst with a “completed” tag. There is a possibility for the analyst to add a completed item into already existing product (selectable from pull-down of pre-existing products, control “add to existing product”), or create a new product pre-initialized to contain this report item (pull down with product types, control “create new product”). The list will only display relevant product types for particular report item type.

Relevant block types will have a spell-check html5 attribute to allow for spell checking. This may be configured on/off on per-user basis, default being yes. Relevant block types will allow to use a pre-defined (canned) text templates. The templates will be global for all users, and separate for each block in the report item. The GUI will allow type-to-search style of canned response lookup.

“Publish” module

This module is used to create a product containing completed report items and publish it to the constituency. There are two different usage scenarios for products:

- a new product can be created in the “publish” module, then populated with report items from a list of finalized report items;
- a new product can be created in the “analyze” module directly from the report item by selecting a product template.

A product can be edited, and its contents displayed (meaning, a list of report items with a brief preview and possibility to jump back to the Analyze module for that report item)

A product can be previewed. This will cause the presenter to be asked to create the presentation of the product, and the result will be available for download via the GUI and if possible, also directly displayed in the GUI. The product will not be published to constituency.

A product can be published via multiple channels:

- send the resulting file via mail
- upload to directory

- upload to CMS
- post on twitter
- upload to MISP

These channels are defined in product configuration.

This will cause the product to be presented via presenter, and irreversibly published via publisher.

“My Assets” module

Module is available for users with “external user” permission, essentially constituents. Asset management allows the external users to:

- upload, review, edit, delete own assets and asset groups (requires asset management permissions)
- add CPE v2.2 and v2.3 codes for my assets
    - by manually selecting an entry from an autocomplete-enabled dropdown
    - by uploading a list of assets together with their CPE codes (v2.2 or v2.3)
- view current products and vulnerabilities relevant for the selected asset
    - filter and sort
- reversed view: my products affected by the vulnerabilities
- configure multiple notification e-mail addresses for each asset group

The module will make use of CPE wildcards - a feature aiming to reduce the number of CPE codes needed to describe a vulnerability. Whenever a CPE code for asset is added that matches a wildcard, that wildcard CPE is also added to such asset. This way, when a wildcard CPE is used in a report item, it will be matched with the asset.

The asset groups and assets will be filtered according to organization, and also will support ACLs.

“Administration” module

The module will provide access to those parts of the system that don’t need to be visited on a day to day basis. This may include settings and system configuration, including:

- user management: ability to change own settings and possibly password (password change is optional, subject to chosen authentication mechanism)
- user management: manage other users and permissions (available with user administrator permission)
- source management: define,review, delete data sources
- remote node management
- audit log: read-only, searchable, exportable (available with audit log permission)
- manage CPE database (upload new version, edit entries, add own entries)
- management of analysis templates and product templates

An analysis template is a named collection of analysis block items in certain order. Analysis block items are detailed in a separate chapter.

Product template is a named configuration that describes a single product type (such as warning-to-mail, bulletin-to-web, etc). Product templates are described in a separate chapter.

Configuration interface for product templates

Product template is a named configuration that describes a single product type (such as warning-to-mail, bulletin-to-web, etc).

The configuration specifies at least the following:

- name of the template
- whether the template supports single report item, or more. Example: to send out single e-mail for each serious vulnerability, the template would be single-item.
- which presenter will be used. Example: PDF presenter
    - and presenter settings, such as title, logo, paper size etc. These differ for each type of presenter. Backend will be able to store the settings in JSON format, and frontend will understand requirements of the existing presenters to display a proper configuration GUI for each presenter type.
- for each report item type that can be included in this particular product, a configuration will be stored, defining how the report item type will be rendered using the presenter, including:
    - what fields from the report item will be rendered and how
- which publisher will be used. Example: SMTP publisher, or CMS publisher
    - and publisher settings, such as list of e-mail addresses, or address+api key+document name within CMS. These differ for each type of publisher. Backend will be able to store the settings in JSON format, and frontend will understand formats of the existing publishers to display a proper configuration GUI for each publisher type.

For e-mail based publisher, the controls for managing the distribution mailing list will include:

- list of e-mail addresses (ordering, search, paging)
- feature to add / remove a single address
- mass import of e-mail addresses. New e-mail addresses may be added in bulk either by pasting the text into a large text field, or by using file upload, with each entry on a separate line. This feature will be purely frontend based, with frontend calling the appropriate number of API calls as if the entries were inserted manually one by one.
- optional signing and/or encryption details for the outgoing e-mail (PGP, S/MIME)

The presenter will modify the content it presents using the following special rules:

- it will be able to filter out some selected sections from the template (please see “marking / highlighting selected parts of the text” in “Analyze” module)
- it will expand CPE wildcard entries to all matched non-wildcard CPEs, if CPE is to be present in the product.

Configuration interface for word lists

The word lists are used for two purposes: input filtering, and highlighting of the received contents. This module will provide a GUI for their configuration.

Word list is structured: each word optionally has a category, and these categories are used for sorting the word list for displaying. Each word also has a flag whether it is a string-based match, or a regular expression. The term “word” is a bit misleading though, because a “word” can also be a phrase (it can contain spaces).

Word lists are one of the entries that are subject to ACLs.

Word lists will be assigned via GUI to certain OSINT sources for filtering, and also used in the Assess module for search and highlighting.

Confguration interface for delegations

There will be a configuration interface with an editable table, where each entry is a combination of: delegation title, a list of e-mail addresses, a list of wordlists can be stored, and an e-mail template. ACL can be optionally applied to this data type.

Configuration interface for OSINT sources, and collectors

The module will provide GUI allowing setup and maintenance of those collectors that are in scope of the project. It will define OSINT sources, and their associated parameters for synchronization. The frontend will allow for setup of all parameters needed by the collector backend.

OSINT source definition GUI will allow for defining the parameters of OSINT source, including keyword lists for filtering.

.

Spec: API and backend requirements

Outline and key points

REST API using external OAUTH authentication, optionally with JWT tokens.

On top of usual RESTful resource-driven URI names, the API will use clearly separated URIs for distinct purposes, so that subsets of the API can be protected by different WAF rules (such as: partner connectors visible from certain IPs or when client certificate is validated; admin part available from LAN; etc.).

If some API calls make sense in multiple contexts, this can be implemented with multiple routes.

/api/*component*/v1/*resource*

where component is one of:

- analysis (for analyst and for wallboards, main API)
- admin (for configuration)
- assets (for external users who maintain their list of assets)
- collector (API exposed to data gatherers)
- connector (API exposed to other parties)
- RESERVED: auth (reserved component name, optionally forwarded to authentication service)

EVERY ACTION that creates, modifies, or deletes any object, or sends out notification, or performs any other such action, will be logged. More detail on logging is provided in the “Security” chapter.

Permissions, ACLs, and access control

Roles

The user may belong to zero or more roles. Roles can be defined on fly. Roles are one of the ways to assign permissions and ACL entries to a group of people in bulk.

Permissions

The system recognizes a set of permissions. Permissions implement Mandatory Access Control security model, by restricting access to API calls, regardless of exact data processed. They are attached to user definable roles, users, or everyone.

The permissions are as fine-grained as possible, so that various workflows, usage patterns and use cases can be accommodated, rule of thumb being: if it is a distinct action, it must have its own permission. That means that most API calls will have a distinct permission, and only a very closely related sets API calls will share common permission.

Security model must be enforced in the backend API. Existence of permission to perform each and every particular action will always be checked server-side. Most of these tests will be performed directly by the framework.

Access control rule: Permissions are checked first, before servicing the API call. IF the user does not have a permission to perform the action directly tied to their user id, AND none of their roles have the permission, AND the permission is not granted to everyone, THEN the access is DENIED.

There is one special permission called “ACL override” that instead of controlling the API calls, allows its user to bypass the ACL DAC model.

Permissions will be configurable via GUI, typically by an administrator with proper permission. In GUI, this is available at a single central place, and within the group and role editors.

ACL entries

The system recognizes ACL entries for certain objects. ACLs implement Discretionary Access Control model by controlling access to processed data, regardless of API call used. Just like permissions, these can be attached to user definable roles, users, or everyone.

Unlike permissions, the ACL entries are only relevant to certain types of data (marked with red border in the entity relationship diagram). Also unlike permissions, if there is no ACL for the data at all, the access is granted.

There are three kinds of ACLs: *see* (will see the entry in a list), *access* (can read all the details, or contents of the entry), and *modify* (can edit, or delete the entry; can add entries inside of this one).

Access control rule: ACL entries are evaluated second, while servicing the API call. IF the user, or some of their roles, or everyone has permission “ACL override”, the access is GRANTED. Otherwise, IF the accessed entry (entry type, and entry id) does not have any associated ACL entries, the access is GRANTED. Otherwise, IF the accessed entry (entry type, and entry id) has defined an access control rule for this user, or any of their roles, or everyone, the access is GRANTED. Otherwise, the access is DENIED.

ACLs will be configurable via GUI, by anyone who is allowed to *modify* the object in question. In GUI, this is available directly where the object is displayed, typically with a “lock” icon that leads to the ACL editor dialog.

Note: ACLs can also be used for limiting access to TLP or security-clearance classified information. By adding the TLP-based ACL to the data, only people with respective TLP role will be able to see the information. For this reason, the collectors will have possibility to apply certain ACLs to the data right at the time of initial upload.

Filtering

Beyond checking the permissions and ACLs before granting or denying the request, certain API calls will further limit access to data depending on the nature of the functionality. For example, the entire asset management API calls will only reveal assets belonging to particular organization, and TaranisNG-to-TaranisNG synchronization will only synchronize and make available the defined subset of data.

Identified required API calls

- CRUD for roles,
- CRUD for assigning roles to users,
- CRUD for permissions,
- CRUD for ACL entries,
- get your own roles, get your own permissions (non-authorized API call)

Identified permissions

- ACL override (a special permission)
- view roles
- edit roles (crud)
- assign roles to users
- view ACL entries
- edit ACL entries

Wallboard and statistics

Supports Wallboard and statistics module.

Identified required API calls

- necessary calls for retrieval of statistics

Identified permissions

- separate permissions for various types of statistics (at least separate privileges to retrieve stats for operators, supervisors, external users and public)

Assess

Supports Assess frontend module.

Identified required API calls

- /analysis/ get list of source groups and number of items in each group
- /analysis/ get list of news items and news item aggregates with attributes and preview (full list or result of search, based on string in title, full text, comments, state), optional ordering and pagination
- /analysis/ controls as specified in frontend description (including upvote/downvote, mark as read/unread, delete, important, send by e-mail)
- /analysis/ aggregate entries/remove selected entries from aggregate
- /analysis/ enable/disable spellcheck (per-user setting)
- /analysis/ retrieve predefined (canned) text templates
- /admin/ CRUD for predefined (canned) text templates

Identified permissions

- view news items
- perform actions news items (multiple permissions with separate delete permission)

Analyze

Supports Analyze frontend module.

Identified required API calls

- /analysis/ CRUD for analysis
- /analysis/ analysis search and list
- /analysis/ CRUD for updating analyze entries of various types, including text, taxonomy entries, file uploads, URIs and other indicators, and more
- /analysis/ change notifications
- /analysis/ get analysis templates aka report item types
- /analysis/ get and set user ordering preferences
- /analysis/ get relevant product types for report item type
- /admin/ CRUD for analysis templates, and contents of analysis templates (list of entry types, their variable names, and their human readable names in particular order; variable names are later used for defining the structure of product to publish)
    - /admin/ CRUD for pre-defined (canned) text values for each separate entry in the analysis template
- /analysis/ API for delegation

Identified permissions

- create analysis
- edit analysis
- view and search list of analyses (for instance search for substrings in title of all open analyses)

Publish

Supports Publish frontend module.

Identified required API calls

- /analysis/ CRUD for product (product contains one/multiple completed report items, based on product type)
- /analysis/ get all completed report items that are not currently assigned to a particular product type, but are suitable for that product type (the product type has a configuration for such report item type)
- /analysis/ create product preview (process product by presenter and store the preview)
- /analysis/ get product preview
- /analysis/ publish product (product presented by a presenter and irreversibly published by publisher)
- /admin/ CRUD for product templates (max. number of report items to include, list of information blocks included report types)

Identified permissions

- create product
- edit product
- preview product
- publish product

My Assets

Supports My Assets frontend module. Supports upload of CPE dictionaries and converts CPE2.2 to CPE2.3 for internal storage. Supports creation of CPE wildcards.

Identified required API calls

- /assets/ CRUD for my assets, CPE assignment
- /assets/ CPE search by substring returning limited set of entries
- /assets/ vulnerabilities for asset
- /assets/ assets for vulnerability
- /assets/ CRUD for custom notifications
- /admin/ CRUD and mass upload for CPE dictionaries, CPE2.2, CPE2.3, CPE wildcards.

Identified permissions

- view my assets
- edit my assets
- update CPE dictionaries
- view CPE dictionaries

Administration

Supports Administration frontend module, and bots (such as a wordlist updater bot).

Identified required API calls

- all API calls required by the frontend
- for analysis template editor, CRUD for templates and their entries
- for product type editor, CRUD for templates
- CRUD for word lists

Identified permissions

- fine grained permissions for each separate administrative action

Collector API and data store

The system needs to be able to receive information about vulnerabilities from various types of feeds. To this end, the system will have an API which will accept connection from various collectors.

The API will also allow the collector modules to store the configuration on a per-instance basis. One part of the configuration will be entered and maintained by the administrator through the frontend web interface (this may be a list of URLs to fetch, regular expressions to parse the output, etc), another part of the configuration will be entered and maintained by the collector itself (hash of the already visited content for deduplication purposes).

Creation and automatic startup of the modules is also a scope of the API backend, but optional

Each collector will connect to the core through API, authenticate either just like a regular user, or using a client certificate (best method to be determined), and upon identification it will receive a configuration that it will use to download its content.

The API will also include support for word lists. These are named collections of keywords and regular expressions. The word lists are per-user and global.

Identified required API calls

- CRUD for collector (define collector, its authentication information, its configuration parameters)
- CRUD for OSINT source (define source, update its fields)
    - comment field: when creating/configuring new sources, user can leave a note for other colleagues why, when and who added/deactivated/edited that source
- CRUD for news item, and a news item attribute. On top of the corresponding permission,

- Store the acquisition date, url-key-value of “seen hashes” with extra ability to delete keys from the store based on
- other calls needed for collectors, as described in a separate chapter
- CRUD for word lists

Identified permissions

- CRUD for collector configuration
- CRUD for data store manipulation (two sets of URLs - one for the front end connections, one for the back end)

Collaboration with remote Taranis NG nodes

Collaboration serves to share the information about news items, and report items, between various Taranis NG teams. It is not meant to synchronize the information to create a single copy of a single data entity, but rather to display the information from remote teams along with the internal information to reduce efforts needed to assess and analyze vulnerabilities and other news items.

For this purpose, the software will allow to configure URLs of the remote node API endpoints (peers), as well as authentication and synchronization parameters. Last synchronization ID will be remembered for each configured peer. It will be sent out as an argument to each subsequent remote call to ensure only changes from the last successful transfer are synchronized. The call will return the changes and a new synchronization ID. The changes that are synchronized are:

- news items (only creation, these are immutable)
- news item attributes (upon creation, and any change)
- report items
- report item attributes

For certain report item attribute types, there will be a special export/import routine to facilitate communication of data that may be unintelligible for the receiving side otherwise. The import/export routines will use common formats for exchanged data, such as: communicate text, not an ID, for any taxonomies; communicate CVSSv3 vector for CVSS calculator, export file attachments as BASE64 encoded string instead of internal links or IDs.

News item synchronization: only

Identified required API calls

- CRUD for remote node
- configure which osint sources are shared with particular node

Identified permissions

- synchronization admin (can configure remote peers to connect to)
- synchronization client (permission for the clients who access the server)

Spec: collectors

Collectors will be separate pieces of software, that will monitor the various news feeds and upload the identified articles using REST API call to the core API.

The following types of collectors will be developed:

- web page collector
- e-mail (POP3, IMAP) collector
- Twitter collector
- RSS collector
- Atom collector
- Slack collector
- Scheduled tasks collector
- Manual entry collector
- out of scope but planned for future: Pastebin collector
- out of scope but planned for future: Microsoft Interflow collector
- out of scope but planned for future: centralized collector for re-use by different Taranis instances

The sources are generally called OSINT sources, and one collector will typically service multiple OSINT sources. Some of the parameters that can be defined for an OSINT source are common, regardless of the collector used, for instance:

    - name and description of the OSINT source,
    - source details: feed accuracy, priority, default confidentiality level, default language
    - comment,
    - source group to put the results into,
    - zero or more filtering word lists. If configured, these will be used to filter the input. Entries not having such keywords will be silently ignored the core API.
    - ACL entries to assign to the source at the time of its creation (optionally based on some condition derived from source text, such as regex match)

The data received from OSINT source contain a list of attributes, some of them derived from the source, some read from the data. These are stored as news item attributes. They may include:

- feed
- accuracy
- priority
- confidentiality level
- language
- author
- timestamp
- tags

ACLs apply for OSINT sources. ACL entries may ALSO and INDEPENDENTLY be configured inside the request which pushes the new news item into the system. This will allow the entries from sensitive sources be created in the system automatically in such way that limits their usage through the system.

See also: "Assess" module.

Web collector

Functionality

Web collector downloads the contents of web pages, including dynamic pages. Based on what the contents of the web page are, it works in two distinct modes:

- direct download
    - the configured URL is considered to be the target page for download.
    - in this case, the collector instance will internally store the hash of the downloaded data, and will compare it to the previously stored value. If the hashes differ, it will push the contents to the API as a new news item.
- linked download
    - the initial URL will be considered an index of articles. Optionally, the configuration will include guide on how to use pagination to search for more results.
    - the source configuration will include a filter that can be used to produce a list of links. Each of these will be visited in the "direct download" mode.

Because the URL of the content may change in the linked download mode, the software will verify whether it has already processed the article based on the content hashes only, not taking the URL into account.

The collector strips the formatting stuff and leaves only the title of the article, and the text of the article. In some cases, two more types of content will be collected as well: attachments, and pictures, associated with the article, author, etc. These are stored as separate news item attributes.

The acquisition must work over the http(s) proxy, or SOCKS proxy, including TOR. The proxy is configured per collector. If the user requires different proxy for various sources, they will use separate collectors for those sources. The acquisition will use a headless browser engine and will allow for basic configuration of the engine, including selection of the user-agent string.

Collectors are separate from the core of the system for security reasons. They may be launched on a separate physical server or VM. Because of this, they only may communicate with the core API via the API calls and must not expect to use other channels such as file system.

Configuration of an OSINT source for web

- for each of the web URLs to be tracked, at least the following information will be stored:
    - “source group” to upload the results to
    - HTTP or HTTPS URL of index page
    - comment
    - time interval for checking
    - optional authentication data (login/password, client certificate)
    - flag: accept invalid certificate (default OFF)
    - optional: regexp or CSS selector for identifying links to follow
    - optional: regexp or CSS selector to identify the “next” pagination button
    - regexp or CSS selector to parse the title or identifier of article
    - regexp or CSS selector to parse the text of article
    - regexp or CSS selector to identify links to the images and attachments to be stored as separate attributes
    - user may optionally configure: source stripping before parsing, for legal reasons: maximum number of characters / words collected
    - user may optionally configure: saving of the author of a document to a separate attribute, mark when the autor is not known (and appropriate selectors)
    - user may optionally configure: selectors to parse identifier of the document, if separate for title

Startup

The collectors may be started automatically by the core API (if the collector runs on the same machine). The collector may also be deployed on a separate server/vm, in which case it is up to the administrator to set up the startup themselves.

GUI based setup of a new web source

In order to allow simple extraction of relevant data from the source, the GUI for new web source will include a selector that will work in a manner similiar to the “web inspector” feature - the user will hover the mouse over the relevant page elements, and the highlighted part will be used as a title, or text to extract. In this mode, the automatically identified CSS selectors will be saved. The GUI must also allow manual configuration with CSS selectors or regexps.

Twitter collector

Twitter collector downloads tweets.

Configuration

- for each of the twitter accounts, the following information should be stored:
    - twitter consumer key, consumer secret, access token, access token secret
- for each of the twitter handles (accounts) to follow, the following may be configured

    - “source group” to upload the results to
    - twitter handle
    - zero or more word lists. If configured, these will be used to filter the input. Entries not having such keywords will be not uploaded to the core API.
- there will also be a possibility to search by hash tags or keywords instead of following an account. In this case, for each of the keywords or hash tags, the same information will be configured:
    - “source group” to upload the results to

Slack collector

Slack collecting bot collects messages via Real Time Messaging API.

Configuration

- for each of the slack accounts, the following information should be stored:
    - slack authentication credentials, API token
- for every slack workspace has to be own instance of collecting bot and should store following information:
    - slack workspace
    - monitored slack channels

Scheduled tasks collector

Scheduled tasks collector is a special kind of collector, that instead of pulling the data from external source, synthesizes the items to the input feed by using predefined data. In "Tasks" there are hourly, daily, weekly tasks which link e.g. external sources the operator should check frequently (like external-systems, wallboard metrics, ticket-system, etc.) and describe working instructions to the operator (restart the client, check fax, write email to a person with usage of pre-created text, etc.) including thresholds. When a threshold is reached / exceeded there are processes / instructions the operator has to follow / to process (e.g. call Person A, write an E-Mail to Person B, etc.).

The information is either pushed to the queue at each and every appropriate time, or - if an optional shell script is defined - the script is launched at that time, and only its successful exit code triggers the push to the queue. Moreover, the title and/or body of the message may refer to special variable ${STDOUT} which will be replaced by the output of the script.

Configuration

The GUI for this particular collector will contain the following configuration for each defined “OSINT Source”:

- title and body of the message to push into the input queue
- cron-like time definition (days of week, hours, dates, months for activation)
- optional script to call. If defined, then the script is launched at pre-defined time and its exit code defines whether the notification will be pushed into the queue.

Manual entry collector

Manual entry collector will be a pure GUI-based implementation that collects the required information through the web form and pushes it inside TaranisNG core.

Spec: presenters and publishers

Presenters

Presenters are separate pieces of software, that convert a set of report items (this will most usually be a description of a vulnerability) to a product (a PDF file, an e-mail, a HTML page, a plain text file, etc.).
The presenter will load its configuration stored at product type to obtain general configuration settings (such as: title of the PDF file, etc.). It will load its configuration stored at product configuration to obtain the information on how to render a particular report item type (which fields to use, etc).
The presenter will return a binary object that can be stored by the main API for the purposes of previewing within the frontend.

Publishers

Publishers are used to irreversibly publish the product by various means (by sending an e-mail, publishing on a webpage via CMS or other) to a particular part of constituency defined in configuration stored at product type.

The following publishers will be developed:

- e-mail publisher
- directory published (uploads the data to a folder)
- CMS publisher for wordpress
- twitter
- MISP publisher (to MISP vulnerability object, https://www.misp-project.org/objects.html#_vulnerability)

Spec: bots

Analyst bot

In its configuration, it has a set of sources it is able to inspect. For each source, it has a list of regular expressions to match, and pre-fills the “analysis” entries from them, including summary, description and recommendations, CVE numbers, CVSS vector and score, URLs, CPE codes.

There will be a special field in the advisory editor, which will be pre-filled by the bot with the original text that was the basis for this output.

Grouping bot

In its configuration, it has a set of sources it is able to inspect. The bot will go over the recently added news items and will try to match them against already downloaded items. If enough similarity is found, the items will be grouped, or an existing group will be expanded.

Wordlist updater bot

A separate bot, that will re-use the same wordlist manipulation API which is used by the GUI.

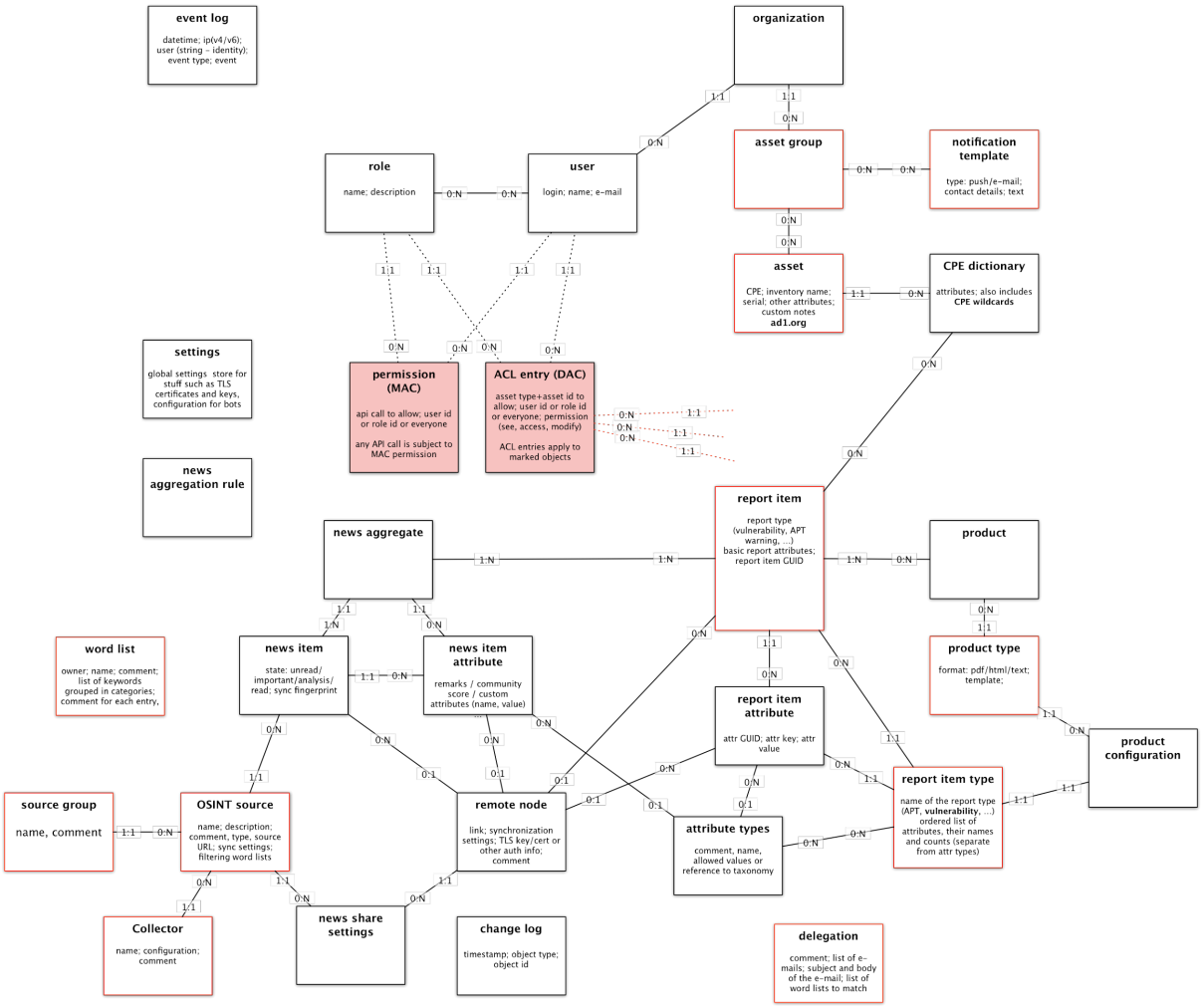
It will allow periodical retrieval of words, and update of the specified word list. Sample use cases: stock companies, government names, etc.

Out of scope: enrichment bot

Future enrichment bot might enrich data on demand / automatically (e.g. APT-Group Synonyms and Description, Malware Descriptions, using e.g. MISP Galaxy). This is not in scope of this project, however the required API calls for this must exist.

Spec: entity relationship diagram

The ER model displayed in this chapter is by no means final. Designers and developers will certainly add other entities, required for successful implementation. The diagram highlights the most important entities and their relations, and selected entity attributes.



Clarification for selected parts of data model

News item and report item attributes

The news item and report item both have a growing list of attributes. In case of the report item, these are pre-filled from a list of attributes appropriate for the report item type when the report item is being created.

The report item types are configured by the administrator of particular Taranis instance, and will vary between the instances. Each report item type (such as: vulnerability report, APT group activity report, etc) has a list of report item attributes. Each of them has a name, and a limit (lower and upper bound, cardinality).

For instance, in a particular configuration, a single vulnerability report item may only have one TLP classification (1-1), at most one “top CVSS” (0-1), but zero or more CVE numbers (0-N) and screenshots (0-N), and a single APT group report will not have any CVSS but needs have one or more APT group identifiers (1-N). It also may have an attachment attribute named “IOCs” (0-N) and another named “malware sample” (0-N). The limits for

report items only apply to internal report item attributes. Remote Taranis instances may chose to publish different attributes and limits for their report item types.

The report item type will also specify ordering of the report item attributes on screen.

Prefered types, formats, contents

This section describes preferred data types and formats for various entities and their attributes.

- sync fingerprint (news item): sha256 or equivalent, made from subset of data that uniquely describe the news item or vulnerability in a manner that is consistent across various instances of Taranis NG, but don't change in course of analysis. For instance, the news item fingerprint might incorporate: OSINT source URL (if available), news item title, news item text. The news item fingerprint will NOT change when the data is subsequently edited, so the same source won't be collected twice.
- GUID(report item): unique identifier of the report item
- (to facilitate synchronization with remote nodes, one of the attribute keys will be FOREIGN GUID; attributes will be assimilated automatically when such attr is added for our own report item)

Technical requirements

- API is auto-started during server boot
- Bots are auto-started during server boot