|  |  |
| --- | --- |
| **Use Case** | User triggers rule update |
| **Actor** | User |
| **Purpose** | Update the central rule database manually |
| **Description** | User sends a command to trigger a rule update, and the system pulls rule files from all the sources specified in the list of sources Rules are then processed and stored in the database. |
| **Pre-conditions** |  |
| **Post-conditions** | Triggers “Update the central rule database from external sources” |

|  |  |
| --- | --- |
| **Use Case** | System triggers rule update |
| **Actor** | System |
| **Purpose** | Update the central rule database automatically |
| **Description** | Configured at intervals, the system is triggered to update and pulls rule files from all the sources specified in the list of sources, processes the rules and then stores them in the database. |
| **Pre-conditions** |  |
| **Post-conditions** | Triggers “Update the central rule database from external sources” |
| |  |  | | --- | --- | | **Use Case:** | Central rule database update | | **Actor:** | User, System | | **Purpose:** | Update the central rule database from external sources | | **Description:** | (1) The case is triggered when either the user issues the command to update, or the system (daemon) issues the command based on a set time-interval. The user has the option to update rules from a local file, if this is the case, the program will proceed from part 3.  (2)  The process starts in the Rule Downloading Module where each rule source is checked for updates by comparing MD5 values. If the MD5 values differ, the module will retrieve the updated rule files and pass them along to the Rule Pre-Storage Processing Module.  (3)The Rule Pre-Storage Processing Module then begins reading the files line by line and parses metadata from the new rule files. The rules are checked against the current rules stored in the database to identify which rules are new or modified. Based on this identification, the new or modified rules are parsed into complete objects and passed to the Database/Core Module that stores the new or modified rules into the database. | | **Pre-condidtions:** | One or more rule sources must be registered in the system. Alternatively, one or more local rule files must be present. | | **Post-conditions:** | Rules have either been confirmed as up-to-date or the new or updated rule objects have been passed to the Database Module. | | |
| **Event flow** | |
| |  |  | | --- | --- | | **User** | **System** | | 1a. User triggers update |  | |  | 1b. System triggers update | |  | 2. Rule Downloading Module compares source with stored rules. | |  | 2a. The rules at source is equal to the stored rules. Database is up to date, so no more action is required. | | 3a. User is notified that database is up to date. |  | |  | 2b. The rules at source is newer than stored rules. Database must be updated. | | 3b. User is notified that the database is not up to date and an update process has begun. |  | |  | 4. Updated rule files are downloaded from sources. The files are validated and passed to the Pre-Storage Processing Module module. | |  | 5. The new files are parsed for rule SID and revision number. | |  | 6. A list of rule SID and revision numbers are gathered from the database and these are compared to identify which rules are new or modified. | |  | 7. New or changed rules are then parsed completely into full objects and these objects are passed to the Database/Core Module. | |  | 8. The Database/Core Module stores the new or changed rule objects into the database. | | 9a. The user is notified that the update is complete. |  | | 9b. The user is notified that rules have been updated during since the last scheduled update interval. |  | | |
| **Feilhåndtering/Alternativer** | |
| 1. The source and database checksum is different but the source rules are older than what is stored in the database.   * Solution: The program can still continue, because if the SIDs already exist they will not be added again and if the revisions are older the rules will also not be updated.   2. Sc | |

**Name:** Rule or rule set is enabled/disabled

**Description:** User sends a command to enable/disable a specified rule set and the system updates the database to reflect that the rule is enable/disabled.

|  |  |  |  |  |  |  |  |  |  |  |  |  |
| --- | --- | --- | --- | --- | --- | --- | --- | --- | --- | --- | --- | --- |
| |  |  | | --- | --- | | **Use Case:** | Rule or rule set is enabled/disabled on one or more sensor | | **Actor:** | User, System | | **Purpose:** | Turn rule sets on or off on demand | | **Description:** | A user wants to either enable or disable a rule or rule set on one or more sensors. This is done by sending a command that specifies which rule or rule set is to be affected, how it is affected and on which sensor or sensors are affected.  The central database must be updated with this new status and the change must be pushed out to the affected sensor or sensors. | | **Pre-betingelser:** | One or more rule sets or rules must be present in the central database. One or more sensor must be registered. | | **Post-betingelser:** | The changes to the rule or rule set must be reflected on the affected sensor or sensors. | |
| **Hendelsesflyt** |
| |  |  | | --- | --- | | **User** | **System** | | 1a. User enables a rule or rule set on a sensor |  | | 1b. User disables a rule or rule set on a sensor |  | |  | 2. The central database changes the status of the rule or rule set by manipulating the status flags. | |  | 3. The system notifies the sensor that a change has occurred. | |  | 4. The change is synced to the affected sensor. | |
| **Feilhåndtering/Alternativer** |
| 1. The change may never be synced to the sensor due to errors.   * Solution: There must be an error protocol that is followed in this scenario. |

**Name:** Treshold is applied/removed to rule

**Description:** User sends a command with either tresholding parameters or removal of threshold for a rule and the system updates the database to reflect the change. One threshold parameter per rule.

**Name:** Suppression is applied to rule

**Description:** User sends a command with either suppression parameters or removal of threshold for a rule and the system updates the database to reflect the change. Suppression parameters can take multiple IP-addresses/IP-ranges.

|  |  |  |  |  |  |  |  |  |  |  |  |  |
| --- | --- | --- | --- | --- | --- | --- | --- | --- | --- | --- | --- | --- |
| |  |  | | --- | --- | | **Use Case:** | Rule is modified/de-modified with a threshold or suppression on one or more sensors | | **Actor:** | User, System | | **Purpose:** | Manipulate configurations for specific rules on specific sensors. | | **Description:** | A user wants to either turn on or off a threshold or suppression modifier on a rule on one or more sensors. This is done by sending a command that specifies which rule the modification will apply to, the parameters of the modification and for which sensor or sensors this will apply for.  The central database must be updated with this new status and the change must be pushed out to the affected sensor or sensors. | | **Pre-betingelser:** | One or more rule sets or rules must be present in the central database. One or more sensors must be registered. If a modification is applied, there must not be another modification for the same rule on the same sensor. If a modification is removed, it must already exist. | | **Post-betingelser:** | The modifications must be applied on the rule and sensor or sensors in question. | |
| **Hendelsesflyt** |
| |  |  | | --- | --- | | **User** | **System** | | 1a. User sets a threshold/suppression on a rule on a sensor |  | | 1b. User turns off a threshold/suppression on a rule on a sensor |  | |  | 2a. The central database creates a threshold/suppression for the specified rule and the specified sensor. | |  | 2b. The central database deletes the threshold/suppression for the specified rule and the specified sensor. | |  | 3. The system notifies the sensor that a change has occurred. | |  | 4. The change is synced to the affected sensor. | |
| **Feilhåndtering/Alternativer** |
| 1. The change may never be synced to the sensor due to errors.   * Solution: There must be an error protocol that is followed in this scenario. |

**Name:** Rules are distributed by the user

**Description:** User sends a command to distribute to specified sensors and the system retrieves rule information for the database, processes them and formats them for SNORT, then distributes them to all sensors specified per their individual settings.

**Name:** Rules are distributed by the system

**Description:** Configured at intervals, the system triggers a command to distribute to specified sensors and the system retrieves rule information for the database, processes them and formats them for SNORT, then distributes them to all sensors specified per their individual settings.

|  |  |  |  |  |  |  |  |  |  |  |  |  |
| --- | --- | --- | --- | --- | --- | --- | --- | --- | --- | --- | --- | --- |
| |  |  | | --- | --- | | **Use Case:** | Rules are distributed | | **Actor:** | User, System, Sensor | | **Purpose:** | Distribute rules from the central database to the local sensors | | **Description:** | A user or the system wants to synchronize the rules stored in the central database to either one, multiple or all sensors. This is initiated either by the user sending a command or by the system having a regular interval.  The system must then calculate and organize the central database data so that the affected sensors get synchronized. The user gets feedback | | **Pre-betingelser:** | One or more rule sets or rules must be present in the central database. One or more sensors must be registered. | | **Post-betingelser:** | The rules on a local sensor must be in sync with the central database. | |
| **Hendelsesflyt** |
| |  |  |  | | --- | --- | --- | | **User** | **System** | **Sensor** | | 1a. User sends a command to synchronize one, more or all sensors |  |  | | 1b. The system reaches an interval for synchronizing all sensors. |  |  | |  | 2. The system calculates and organizes the data for each sensor affected by the synchronization. |  | |  | 3. The system initializes a synchronization protocol with the affected sensor or sensors, sending a list of what the sensor should have. |  | |  |  | 4. The sensor calculates diff between should have and has. | |  |  | 5a. The sensor content is different from what it should have, asks central system to send what it needs. | |  |  | 5b. The sensor content is not different from what it should have, no further action is required. This status is sent to the central system. | |  | 6a. The system retrieves and sends what the sensor asks for. |  | |  | 6b. The system terminates the synchronization protocol with the sensor in question. |  | |  |  | 7. The sensor includes the new changes into its database and files. Tells the central system that it is now up to date. | |  | 8. The system carries out maintenance to reflect the synchronization. |  | | 9. The user is notified that all sensors affected are now up to date. |  |  | |
| **Feilhåndtering/Alternativer** |
| 1. The change may never be synced to the sensor due to errors.   * Solution: There must be an error protocol that is followed in this scenario. |

**Name:** Viewing rules

**Description:** Rules are retrieved from the database, processed for GUI/CLI format and then delivered to GUI or CLI in specified format.

|  |  |  |  |  |  |  |  |  |  |  |  |  |
| --- | --- | --- | --- | --- | --- | --- | --- | --- | --- | --- | --- | --- |
| |  |  | | --- | --- | | **Use Case:** | Rules and information about rules are presented to a user | | **Actor:** | User, Presenter, System | | **Purpose:** | So the user can get a human readable overview of the rules and manipulate them. | | **Description:** | A user wants an overview of the rules stored in the central database, the status of these rules in relations to the sensors and various meta data related to both rules and sensors.  This will be presented in some form of graphical list or lists where the user can manipulate the rules in various ways. | | **Pre-betingelser:** | The presenting sofOne or more rule sets or rules must be present in the central database.tware must be running and answering requests from the user. | | **Post-betingelser:** | The user is shown the current status of rules and sensors. | |
| **Hendelsesflyt** |
| |  |  |  | | --- | --- | --- | | **User** | **Presenter** | **System** | | 1. User brings up overview of rules. |  |  | |  | 2. The Presenter asks the System for data, limits it to the current subset of rules needed to present to the user and then some extra for caching. |  | |  |  | 3. The system calculates and retrieves the necessary data, hands it back to the Presenter. | |  | 4. The Presenter populates the users interface with the data, then reverts to step 2 whenever the user asks for different data. |  | | 1. User manipulates a rule. |  |  | |  | 2. The Presenter sends the changes to the System. |  | |  |  | 3. The System processes the change and stores the result. The result is also sent to the Presenter. | |  | 4. The Presenter shows the result to the User. Asks if the user wants to commit to sensors. |  | | 5a. The user does not want to commit now. |  |  | | 5b. The user wants to commit to sensor now. |  |  | |  | 6. The Presenter passes this choice to the System. |  | |  |  | 7a. The system stores the change for a future synchronization. | |  |  | 7b. The system synchronizes the change to the affected sensors. | |  |  |  | |
| **Feilhåndtering/Alternativer** |
| 1. The change may never be synced to the sensor due to errors.   * Solution: There must be an error protocol that is followed in this scenario. |

**Name:** User comments on rule or rule set

**Description:** User specifies a rule or rule set and applies a comment to it, that is then stored in the database.

**Name:** User edits/deletes a comment on a rule or rule set

**Description:** User specifies a rule or rule set and edits or deletes it from the database.

|  |  |  |  |  |  |  |  |  |  |  |  |  |
| --- | --- | --- | --- | --- | --- | --- | --- | --- | --- | --- | --- | --- |
| |  |  | | --- | --- | | **Use Case:** | User wants to add a comment to a rule or rule set | | **Actor:** | User, System | | **Purpose:** | Adding user comments to a rule or rule set. | | **Description:** | A user wants to add a comment to a rule or rule set, in order to store meta information about them, such as critique. | | **Pre-betingelser:** | One or more rule sets or rules must be present in the central database. | | **Post-betingelser:** | A comment has been attached to a rule or a rule set. | |
| **Hendelsesflyt** |
| |  |  | | --- | --- | | **User** | **System** | | 1. User sends a command with comment data. |  | |  | 2. The system stores the comment and attaches it to the rule or rule set in question. | |
| **Feilhåndtering/Alternativer** |
| 1. The change may never be synced to the sensor due to errors.   * Solution: There must be an error protocol that is followed in this scenario. |

**Name:** User registers a sensor

**Description:** User sends parameters for a sensor and the system stores the parameters in the database.

**Name:** User registers a source for rule updates

**Description:** User sets parameters for sources and the system stores the parameters in the database.

**Name:** System verifies rules as valid and functional

**Description:** Before rules are stored in the database, the system process and check them for validity and functionality by doing a test run in a local SNORT to catch any errors they might produce and register in the database

**Name:** User organizes sensors into sensor group

**Description:** A user sets up a sensor group and then includes one or more sensors into this group. The user can then manipulate rules on a sensor group basis.

**Name:** d

**Description:** d

**Name:** d

**Description:** d

**Name:** d

**Description:** d

**Name:** d

**Description:** d