Developer Guide for Nexpose Ticketing System

Contents

Last Revised: November 19, 2015

RAPID7

Solution Summary

The goal of incident management is to restore normal service operation as quickly as possible following an incident, while minimizing impact to business operations and ensuring quality is maintained. The integration with Nexpose allows customers to create incident tickets based on vulnerabilities found across their systems. Using the Nexpose Ticketing Gem will allow users to implement their own Integration endpoint within the ticketing system.

Before You Begin

The integration is written in Ruby and therefore requires Ruby binaries to be installed on the host system. The following link shows the different options for installing Ruby in several platforms:

- https://www.ruby-lang.org/en/downloads/

Once installed, the next step would be to check out the source code from GitHub, located at the following address:

- https://github.com/rapid7/nexpose_ticketing

Source Code Directory Structure

The source code has the following structure:

```
\---nexpose_ticketing
   | .gitignore                                    gitignore File.
   | nexpose_ticketing.gemspec                     Gemspec the ticketing system.
   | README.md                                     README File.
   | Gemfile                                       Gemfile.
   |
   \---bin
   |   | nexpose_jira                              JIRA implementation.
   |   | nexpose_remedy                            Remedy implementation.
   |   | nexpose_servicedesk                       ServiceDesk implementation.
   |   | nexpose_servicenow                        ServiceNow implementation.
   |
   \---lib
   |   | nexpose_ticketing.rb                      Main interface of the service.
   |   |
   |   \---nexpose_ticketing
   |     | queries.rb                              SQL queries.
   |     | common_helper.rb                        Common methods used by the service
   |     |                                         helper classes.
   |     | ticket_repository.rb                    Report generator.
   |     | ticket_service.rb                       Ticketing service.
   |     | nx_logger.rb                            Log service.
   |     | report_helper.rb
   |     |
   |     \---config
   |       | jira.config                           JIRA configuration file.
   |       | remedy.config                         Remedy configuration file.
   |       | servicedesk.config                    ServiceDesk configuration file.
   |       | servicenow.config                     ServiceNow configuration file.
   |       | ticket_service.config                 Ticket service config.
   |       |
   |       \---remedy_wsdl
   |         | HPD_IncidentInterface_Create_WS.xml
   |         | HPD_IncidentInterface_WS.xml
   |       |
   |       \---servicenow_updateset
   |         | Rapid7 Nexpose Ticketing.glide-calgary-02-15-2013__patch2-hotfix5.xml
   |     |
   |     \---helpers
   |       | jira_helper.rb                        JIRA helper implementation.
   |       | remedy_helper.rb                      Remedy helper implementation.
   |       | servicedesk_helper.rb                 ServiceDesk helper implementation.
   |       | servicenow_helper.rb                  ServiceNow helper implementation.
   |
   \---test                                        Test files below.
   |   | data_report.csv
   |   | empty_report.csv
   |   | nexpose_ticketing_spec.rb
   |   | sample_historical_scan_data.csv
   |   | ticket_repository_spec.rb
   |   | two_vulns_report.csv
   |   |
   |   \---helpers
   |     | jira_dummy_one_ip_ticket.txt
   |     | jira_dummy_two_default_tickets.txt
   |     | jira_helper_spec.rb
   |     | +++++++
```

Example of a New Implementation - JIRA

Any new implementation should have three important files:

1. **Helper File:** This file should implement the methods to transform a CSV report into the appropriate format. For example, JIRA expects a JSON object for the ticket format.

2. **Config File:** The configuration file defines implementation specific details such as username, password and address needed by the helper to create a ticket. NOTE: The field marked as 'helper_name' is mandatory and should have the name of the Helper File class. This is read at service startup.

3. **Executable File:** This file should read the config file and call the main NexposeTicketing main class with the configuration options.

JIRA Implementation

In the case of the JIRA implementation, the three files are explained below:

1) **Helper File: jira_helper.rb:**

This file implements three main methods which are necessary to be able to create tickets:

1. Initialize: This is the constructor that will take the implementation options and the service options.

2. create_ticket (tickets): This method will take in an array of tickets previously created and "send" them to the ticketing system. In the case of JIRA, the transport is through a HTTPS POST method, using the data from the jira.config file, sending the tickets to the user specified URL.

3. prepare_create_tickets (vulnerability_list, nexpose_identifier_id): This method will prepare tickets for sending to the JIRA service, parsing the CSV report (vulnerability list) to the format the ticketing system uses. In the case of JIRA this format is in JSON.

   The common_helper.rb class has methods to generate a NXID (unique identifier for a ticket. For more information, please see the Nexpose Ticketing Configuration Guide) from the provided nexpose_identifier_id. If implementing the helper to work in the different ticketing modes, the CSV vulnerabilities should be grouped into individual tickets as follows:

   a. The 'Default' mode means that there should be a ticket per IP and Vulnerability. For example, if on a scan of IP 1 we find vulnerability A, B, C. Three tickets are created: IP 1 -> A, IP 1 -> B, IP 1 -> C.

   b. In IP mode we are grouping by individual IPs. Using the previous example there would only be one ticket created: IP 1 grouping A, B, C into one ticket.

   c. For Vulnerability mode we are grouping by the individual vulnerabilities. In the example there would be three tickets created, one for each vulnerability containing one IP in each.

   This 'matching fields' value is used to group related vulnerability information together when creating tickets. For each row corresponding to a new ticket in the CSV file, a new ticket object will be created with the correct values and description. For every subsequent row that is part of the same ticket, the ticket description will be updated by the common_helper.rb class. When a new matching field value is encountered, the previous ticket is converted to a JSON object and placed in the array of created tickets, before creating a new ticket for the current row. This array of tickets is the returned to the parent method.

   Please refer the jira_helper.rb file for an example of the implementation.

4. To implement the full functionality of the ticketing service, the following methods should also be implemented:

   a. prepare_update_tickets (vulnerability_list, nexpose_identifier_id): This method prepares ticket updates from a provided CSV list of vulnerabilities. JIRA uses a ticket key to push updates, so first all the tickets are prepared using the prepare_create_tickets method, allowing new tickets to be sent along with the updates. For each created ticket the method then tries to get the corresponding JIRA key using get_jira_key. This key, either as a value or nil, is put into a key value pair along with the generated ticket. This pair is then appended to an array of tickets to send. The nil value allows the update_ticket method correctly differentiate between new and existing tickets, allowing it to send the whole ticket when it does not yet exist. This method is used by the ticketing helper to update existing tickets in JIRA when in IP and vulnerability mode

   b. update_tickets (tickets): This method is for sending ticket updates in JSON format to the JIRA service. Each ticket in the provided list is a key value pair representing its JIRA key and the corresponding newly generated ticket description. If a ticket is to be updated, then the JIRA key is appended to the URL, and the request body contains an update message in JSON format containing the new description for the ticket. This is then pushed to the service. If the ticket does not yet exist (no corresponding JIRA key) then this is considered a new ticket. The request body instead contains the whole ticket, which is then sent to the service. These tickets are sent as individual HTTP POST requests, using the settings and user specified URL from the jira.config file.

   c. prepare_close_tickets (vulnerability_list, nexpose_identifier_id): This method uses a provided CSV file to query JIRA for the corresponding JIRA key. For each row in the file the method calls get_jira_key, passing in a query containing the project number and the nxID of the issue. The returned key is placed in an array with its corresponding ticket before being returned.

   d. close_tickets (tickets): This method is for sending ticket closure messages in JSON format to the JIRA service. A list of valid JIRA ticket keys are provided to the method, with each key representing a ticket completed before the last scan. For each ticket, the method calls get_jira_transition_details, requesting the steps to get from the tickets current state into the user selected 'closed' state. A HTTP request is then sent to JIRA to transition the selected ticket into the 'closed' state.

2) **Configuration File: jira.config**

The configuration file defines the entire information specific with the Helper class. The jira.config file has:

```
# (M) Helper class name.

:helper_name: JiraHelper

# Optional parameters, these are implementation specific.

:jira_url: https://url/rest/api/2/issue/

:username: jirausername

:password: jirapassword

:project: projectname
```

The **mandatory** option is the class name for the helper under helper_name. This name is used to load dynamically the developed helper. The other fields are implementation specific, in the case of the JIRA example, we require a URL, username, password and project. Remember, these variables are only used by the Helper file you develop so they are very implementation specific.

3) **Executable File: nexpose_jira**

The executable file parses the configuration file, parses it into a Hash of variables and then initializes the ticket service with the data calling the method start: NexposeTicketing.start (jira_options).

Please refer to the Executable file nexpose_jira under the bin folder for an example.

Packaging it Up

Where the files are saved is very important; once the gem is installed, any development done should be saved to the specific folders:

- Executable file: The executable file should be saved to the bin folder.
- Helper file: All helper files reside under the /lib/helper directory and the service expects to load the helper file from there.
- Configuration file: The configuration file should reside under /config folder.

All these folders can be found usually under the following paths:

- Windows: C:\Ruby<version>\lib\ruby\gems\<version>\gems\nexpose_ticketing\lib\nexpose_ticketing\
- Linux: /var/lib/gems/<version>/gems/nexpose_ticketing/lib/nexpose_ticketing/

Your installation folder may differ, please refer to the Ruby documentation for the specific location.

Initial Run

Assuming you've properly configured the Nexpose and your helper parameters, type in the executable file created from a command/bash shell.

Every time this command is run the service will query Nexpose and obtain any new vulnerability information and open tickets accordingly.

Troubleshooting

The most common errors when running the script are:

- Configuration errors (such as incorrect indentation or user details).
- Specifying a user without sufficient permissions to create tickets.
- Specifying a Nexpose user without permission to create reports.
- Not specifying a site or tag.
- Specifying both a tag and a site. The tag will take priority and the site will not be scanned.

We recommend reading the log file under the log folder in the Gem - ensure to log information at important points in your custom helper to enable troubleshooting during development.

If everything still fails, please send an email to integrations_support@rapid7.com with the ticketing_service.log attached and a description of the issue.

Common Helper Class

The CommonHelper class, found in lib/nexpose_ticketing, is used by the service specific helpers to perform common tasks. These methods can be used to influence functionality in new helpers. Please see current helper classes for how and where these methods can be implemented when creating a new helper. Below is an overview of the methods:

- **get_description** - Common method to generate a ticket description with the provided data. Passes the parameters to a mode-specific method for generating the description.
- **update_description** - Common method to update a ticket description for an existing ticket. Passes the parameters to a mode-specific update method for the current ticketing_system mode. This method is called to insert data from subsequent rows in the generated Nexpose CSV file, adding the information to the description. This mode only applies changes in IP and Vulnerability mode.
- **print_description** - Converts the description hash object into a string, appending the NXID. Selects the correct method for the current mode.
- **get_default_ticket_description** - This method is used to generate the description for a Default ticket from a row of data from the Nexpose query CSV file. It calls the get_vuln_header and get_discovery_info methods to create the ticket header, get_references to generate the references and get_solutions to generate the solutions.
- **get_ip_ticket_description** - This method is used to generate the description for an IP ticket from the initial row of data from the Nexpose query CSV file for a specific IP. Vulnerabilities are sorted under 'New', 'Same' and 'Old' headers based on the vulnerability status since the last scan. The information for the initial vulnerability is generated using the method get_vuln_info and appended to the end of the description.
- **get_vuln_ticket_description** - This method is used to generate the description for a Vulnerability ticket. The header is generated using the get_vuln_header method, get_references for the vulnerability references, the solutions for the vulnerability with get_solutions and get_assets returns the assets affected by the vulnerability in this row.

- **update_ip_ticket_description** - This method is used to add subsequent rows from the CSV file, listing more vulnerabilities affecting the current IP. If the status of the vulnerability is the same as the previous row (New / Same / Old), then the vulnerability is appended directly to the description, with the vulnerability description generated by the get_vuln_info method. If however, the status differs, then a header is inserted before the vulnerability, stating the status of this added vulnerability.

- **update_vuln_ticket_description** - This method is used to add assets from a different row of the CSV file to the current list of assets affected by this vulnerability. These assets will have a different status to the original assets, hence why they require a separate row - for instance the initial row may be for assets which had this vulnerability during the last scan, the next row could then be assets which used to have this vulnerability, or assets which no longer have this vulnerability. The assets are added to the descriptions assets using the get_assets method, passing the current row.

- **print_default_ticket_description** - This method formats the data collected from the CSV file for insertion into a Default mode ticket description - beginning with the vulnerability header, followed by the references for the vulnerability, with the solutions below.

- **print_ip_ticket_description** - This method formats the data collected from the CSV file for insertion into an IP ticket description - each value in the vulnerability array is appended to the ticket description. Each vulnerability was correctly formatted in the get_vuln_info method (given a header, listing references and solutions) and so it is appended directly onto the end.

- **print_vuln_ticket_description** - This method correctly formats data collected from the CSV file for insertion into a Vulnerablitily ticket description - the header is placed at the top, followed by a list of the assets affected by this vulnerability, each with a header stating if this asset had this vulnerability in the last scan, or if it was only discovered to have it in the most recent scan, as stated in the update_vuln_ticket_description method. This is then followed by the references for the vulnerability and the solutions.

- **generate_nxid** - Used by the helper class to generate the unique NXID based on the chosen mode. Used to identify individual tickets when creating, updating and closing tickets.

- **get_vuln_info** -Formats the row data for a single vulnerability to be inserted into a Default or IP mode ticket. This will be called once for each Default mode ticket, and as many times as there are vulnerabilities for an IP ticket. At the top of the vulnerability description is the header, created with the method get_vuln_header. After this is inserted the discovery information, with get_discovery_info. Then the references for the vulnerability are inserted with the get_references method. Finally the solutions for the vulnerability are inserted using get_solutions.

- **get_vuln_header** - Generates the common vulnerability header for each ticketing mode. This is placed at the top of each Default and Vulnerability ticket, and placed above each different vulnerability in an IP ticket.

- **get_title** - This method is called from the chosen ticketing system helper to generate the title for a ticket, based upon the current ticketing mode. In Default mode the title is the IP of the machine, and the short summary of the vulnerability, generated using the get_short_summary method. For IP mode, it is the IP of the machine and the header "Vulnerabilities". For Vulnerability mode the title is the short summary of the Vulnerability taken directly from the CSV file. This title variation makes it simple to differentiate between different types of tickets.

- **get_short_summary** - This method is used to generate a short summary for a vulnerability from a row in the CSV file. It takes the solution description from the file, and takes the first part as the short description of the vulnerability for insertion into a ticket.

- **get_solutions** - This method is used in all ticketing modes to get and format the solutions for a vulnerability for insertion into a ticket. It substitutes the solution delimiter for a new line character, separating them when inserted.

- **get_discovery_info** - This method is used to insert the discovery information about a vulnerability into a ticket - when it was first discovered, and when it was last seen - this is inserted after the vulnerability header.
- **get_references** - Formats the references for a vulnerability in into a format suitable to be inserted for insertion into a ticket. Used by all modes.
- **get_assets** - Returns the assets for a vulnerability in a format suitable to be inserted into a ticket. Used by Vulnerability mode when formatting row information. Groups the assets based on when the vulnerability was detected - in a previous scan, the most recent scan, or when it is no longer present.
- **get_field_info** - This method is used to return the information from a row which is used as the matching field for the ticketing mode to enable grouping tickets on the matching field - for IP mode it will return the assets IP address, for vulnerabilities it will return the vulnerability ID and for default mode it will return both the vulnerability ID for the current vulnerability and the IP address of the asset.

Queries

Queries are defined in the queries.rb file and are selected on the basis of ticketing mode, taking into account whether there has been a pre-existing scan. When designing new queries, please use the existing queries as templates.

There are several main queries used by the ticketing system:

Query	Description
last_scan	Gets all the latest scans for sites.
last_tag_scan	Gets all the latest scans for tags.
all_new_vulns	Gets all delta vulns for all sites sorted by IP.
all_new_vulns_by_vuln_id	Gets all delta vulns for all sites sorted by vuln ID.
new_vulns_since_scan	Gets all new vulnerabilities happening after a reported scan id.
new_vulns_by_vuln_id_since_scan	Gets all new vulnerabilities happening after a reported scan id. Sorted by vuln ID.
old_vulns_since_scan	Gets all old vulnerabilities happening after a reported scan id.
all_vulns_since_scan	Gets all vulnerabilities happening after a reported scan id. This result set also includes the baseline comparison ("Old", "New", or "Same") allowing for IP-based ticket updating.
all_vulns_by_vuln_id_since_scan	Gets all vulnerabilities happening after a reported scan id. Sorted by vuln ID. This result set also includes the baseline comparison ("Old", "New", or "Same") allowing for IP-based ticket updating.
old_tickets_by_ip	Gets all IP addresses that have only old vulnerabilities i.e. any open tickets can be closed.
old_tickets_by_vuln_id	Gets all old vulns that have no active IPs i.e. any open tickets in vuln mode ('V') can be closed.