Custom Vulnerability Checks with QRDI

User Guide

August 14, 2017

Qualys, Inc.
1600 Bridge Parkway
Redwood Shores, CA 94065
1 (650) 801 6100

Contents

About this Guide

Thank you for your interest in the Qualys Cloud Platform and custom vulnerability checks! This guide tells you how to add custom vulnerabilities using Qualys Remote Detection Interface (QRDI) and execute them by launching Qualys Vulnerability Management scans.

About Qualys

Qualys, Inc. (NASDAQ: QLYS) is a pioneer and leading provider of cloud-based security and compliance solutions with over 9,200 customers in more than 100 countries, including a majority of each of the Forbes Global 100 and Fortune 100. The Qualys Cloud Platform and integrated suite of solutions help organizations simplify security operations and lower the cost of compliance by delivering critical security intelligence on demand and automating the full spectrum of auditing, compliance and protection for IT systems and web applications. Founded in 1999, Qualys has established strategic partnerships with leading managed service providers and consulting organizations including Accenture, BT, Cognizant Technology Solutions, Fujitsu, HCL Comnet, HPE, Infosys, NTT, Optiv, SecureWorks, Tata Communications, Verizon and Wipro. The company is also founding member of the Cloud Security Alliance (CSA). For more information, please visit www.qualys.com

Qualys Support

Qualys is committed to providing you with the most thorough support. Through online documentation, telephone help, and direct email support, Qualys ensures that your questions will be answered in the fastest time possible. We support you 7 days a week, 24 hours a day. Access support information at www.qualys.com/support/

Get Started

Now you can easily add custom vulnerabilities (QIDs) using Qualys Remote Detection Interface (QRDI) and execute them by launching Qualys Vulnerability Management (VM) scans via the Qualys Cloud Platform UI and API.

For each QRDI vulnerability you'll provide:

- vulnerability settings similar to Qualys provided vulnerability (i.e. title, category, severity, threat, impact, solution, mappings), and
- QRDI definition, in valid JSON format, that describes the logic of the vulnerability detection (simple HTTP requests are supported)

QRDI detection results:

- the output of a QRDI vulnerability detection is similar to any Qualys provided vulnerability detection, i.e. QID instances appear in scan reports, API output, asset information etc. in the same way.
- scan processing does not occur when QRDI vulnerability has debug mode enabled. When debug mode is enabled detection results will appear in scan results, but will in template based scan reports, and any ticketing rules are not triggered.

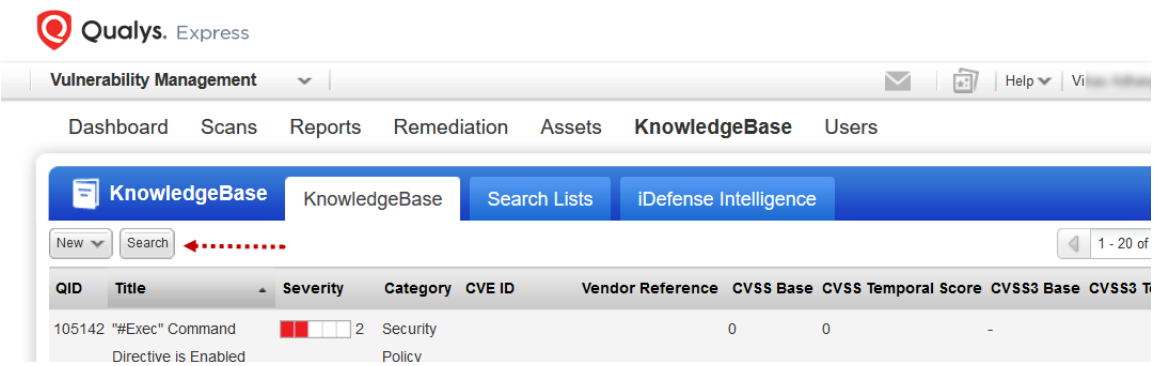
What you'll need

Qualys Cloud Platform account with Qualys Custom QRDI Checks enabled. Manager role is required to create and edit custom QRDI checks. Need some help with this? Just reach out to Qualys Support or your Qualys Account Manager.

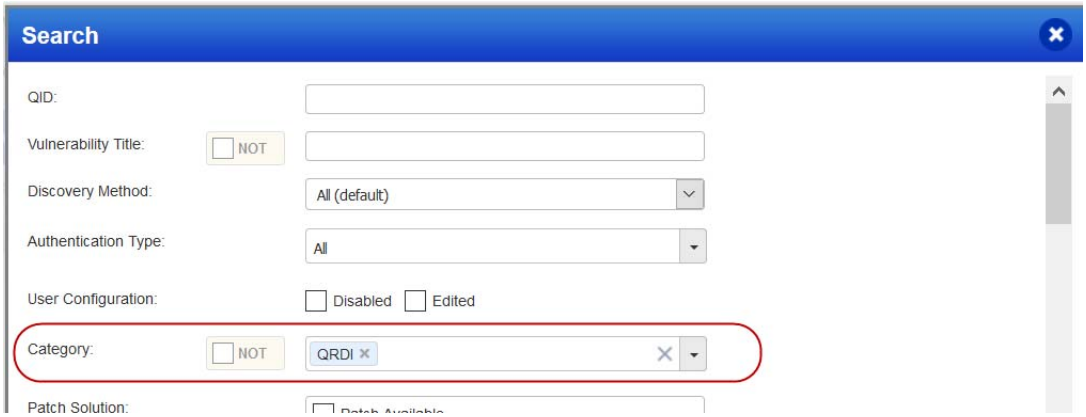
Let's have a look at QRDI vulnerabilities

Once you've added custom QRDI vulnerabilities they appear in the KnowledgeBase section in your account, just like Qualys provided vulnerabilities.

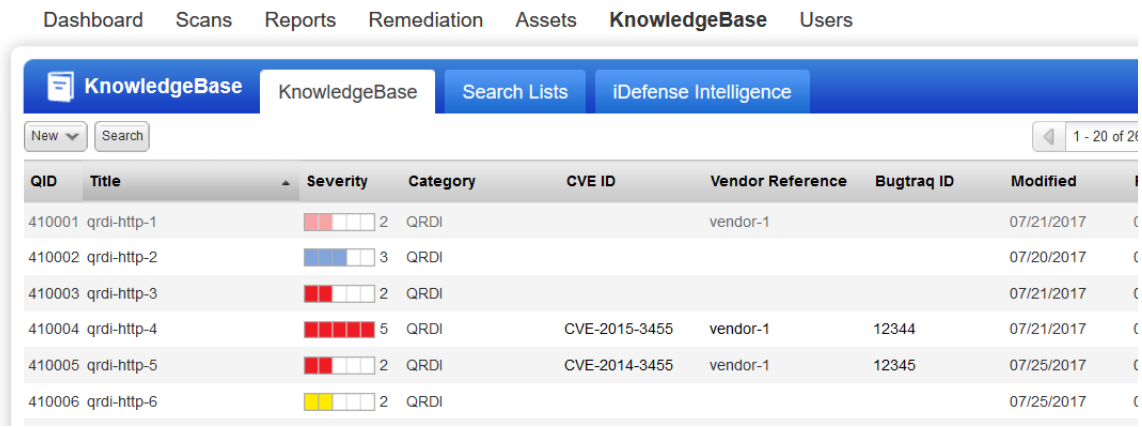
Click the Search option to quickly find all your custom QRDI vulnerabilities:



Choose the category QRDI for your search:

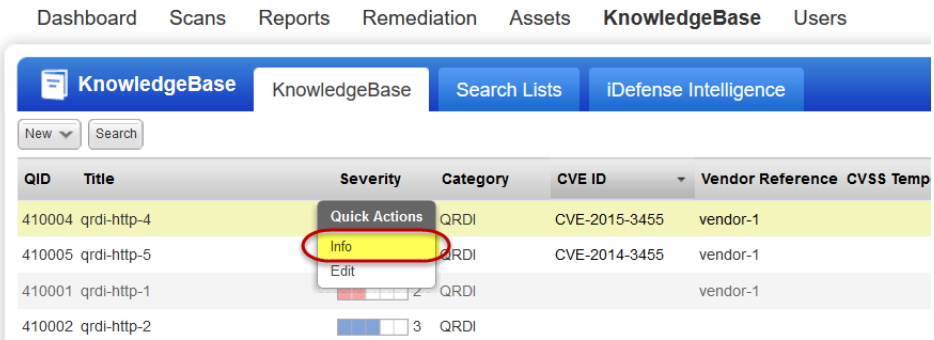


Here's a sample list of QRDI vulnerabilities:

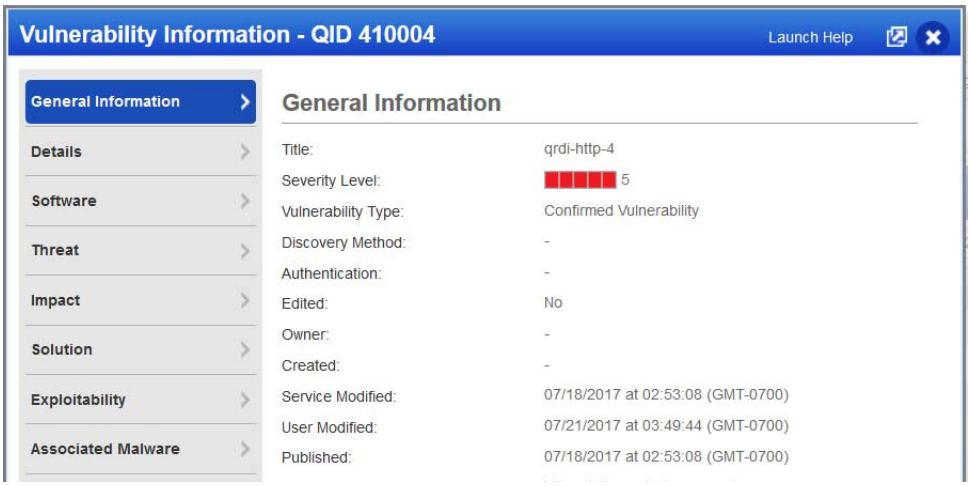


Drill down to vulnerability info just like any Qualys provided vulnerability:

Hover over the QID of interest and select
Info from the Quick Actions menu



View vulnerability info and edit settings as needed:



Common Questions

How many QRDI vulnerabilities can I add?

- You can add a maximum of 20,000 QRDI vulnerabilities to your subscription.

What about the vulnerability settings?

- When you add a QRDI vulnerability, you'll provide title, severity, type, descriptions, optional mappings (CVE IDs, Bugtraq IDs, vendor references) and the QRDI definition in the form of a JSON document. The category is always QRDI. Tip - You can search the KnolwedgeBase for category QRDI to find your custom QRDI vulnerabilities.

Tell me about QIDs for QRDI vulnerabilities

- Qualys assigns QIDs to new QRDI vulnerabilities automatically in sequential order in the range 410,000-430,000. The first QRDI vulnerability you add will have the QID 410,000, the second will have QID 410,001, and so on.

Can I edit a QRDI vulnerability?

- Sure you can edit the general information (title, severity, descriptions, mappings etc.) and the QRDI definition (JSON document) at any time. Note you can't edit these settings: the category QRDI, and the vulnerability type if it's Information Gathered.

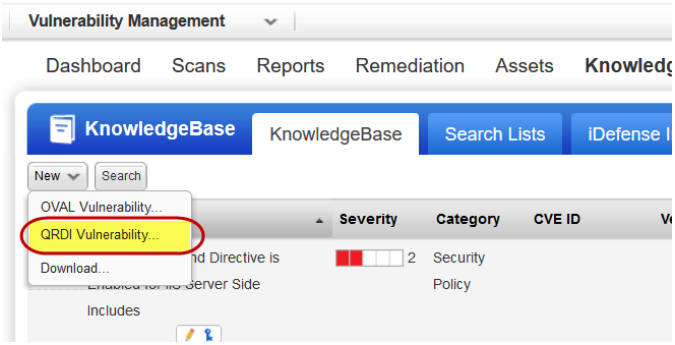
Can I remove a QRDI vulnerability

- It's not possible to remove/delete the QID for QRDI vulnerabilities once added to your subscription.

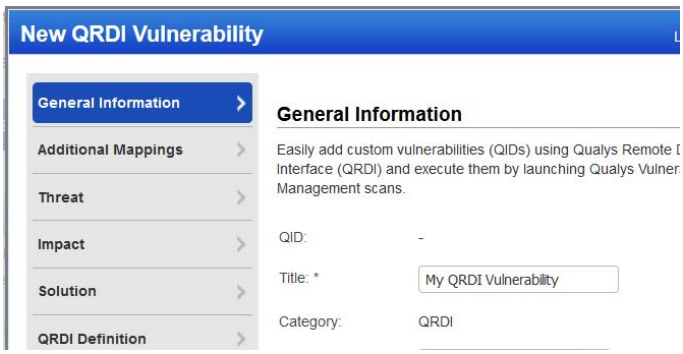
Is there support to enable/disable QRDI vulnerabilities?

- Yes you can enable/disable a QRDI vulnerability just like a Qualys provided vulnerability by editing the vulnerability settings. Good to Know - When a vulnerability is disabled and it is included in the scan settings (i.e. option profile), it will be scanned like any other vulnerability and it will appear grayed out in scan results and reports.

How to add QRDI vulnerabilities



Go to KnowledgeBase and select New > QRDI Vulnerability.

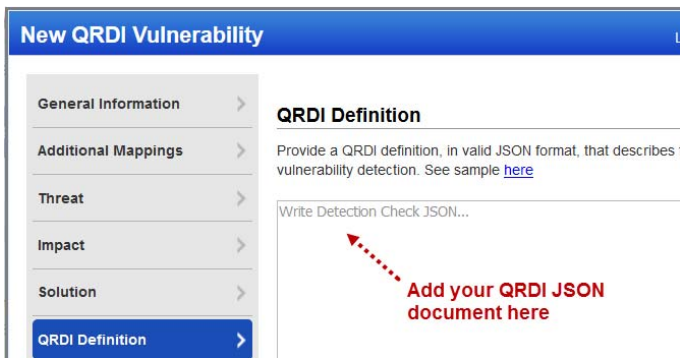


Provide QRDI vulnerability settings:

- general information (title, type, severity)

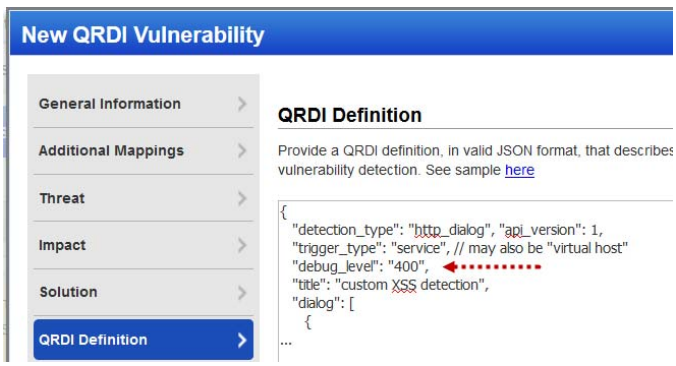
- optional mappings (CVE IDs, Bugtraq IDs, vendor references)

- descriptions for threat, impact, solution



Upload a JSON document that describes the logic of the detection, i.e. the conditions under which to run the detection, the sequence of data to be sent and received, any pattern matching rules, other data required to run the detection.

Learn more



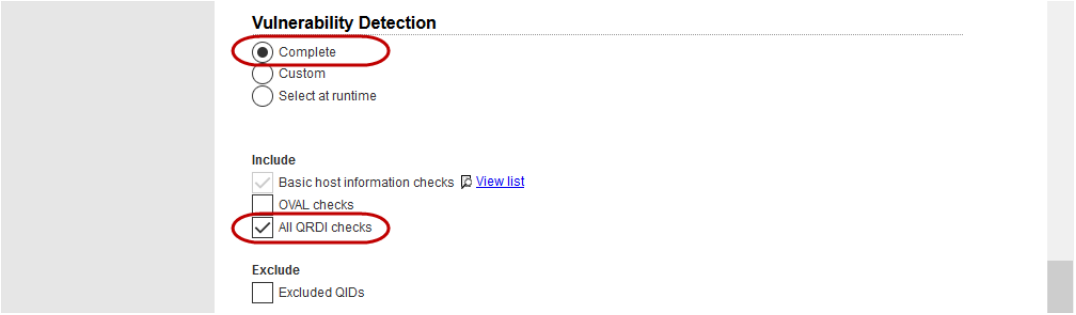
We recommend debug mode is enabled for the QRDI vulnerability.

To enable debug mode, add “debug_level” in the top-level JSON object and a value that indicates verbosity of logging info (least info to most info): 100, 200, 300 or 400

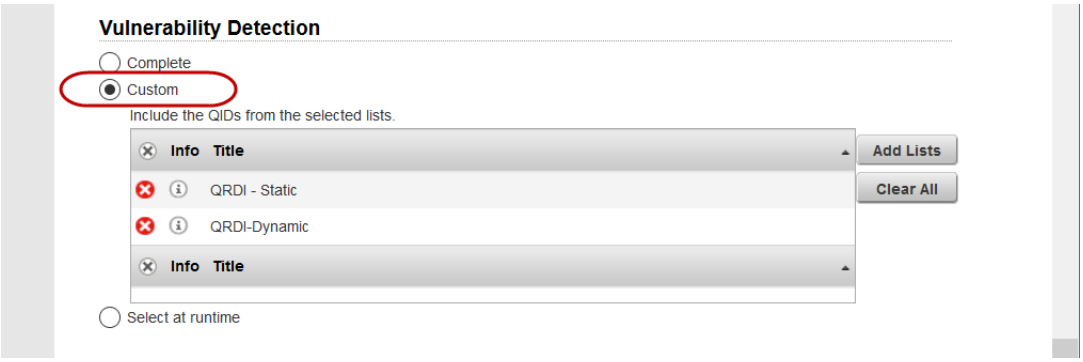
How to configure scan settings

Customize the Vulnerability Detection section in the option profile you'll use for scanning. There's a few ways to do this.

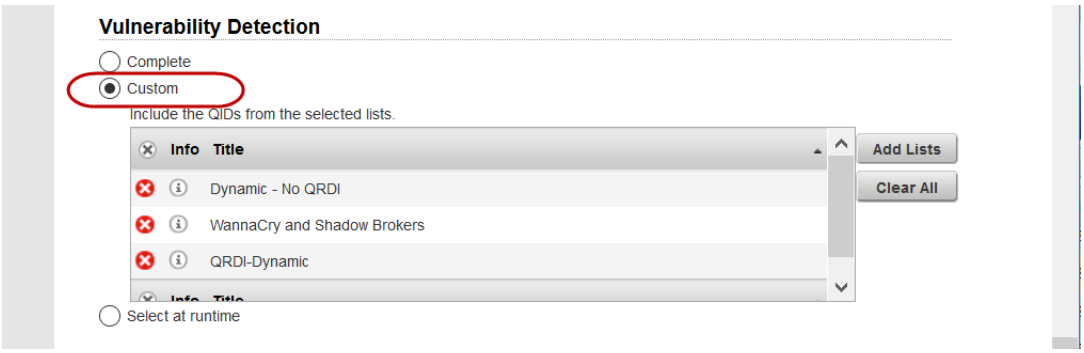
Option 1 - Scan for all vulnerabilities in your account - This includes Qualys provided vulnerabilities plus all QRDI vulnerabilities. Select "Complete" and "All QRDI" checks as shown.



Option 2 - Scan for selected QRDI vulnerabilities. Create search lists (static and/or dynamic) including only QRDI vulnerabilities, select "Custom" and add your custom lists. In this sample QRDI vulnerability checks matching a static search list and a dynamic search list are selected.



Option 3 - Scan for selected vulnerabilities - Qualys provided and QRDI. Create search lists (static and/or dynamic) including QRDI vulnerabilities plus Qualys vulnerabilities, select "Custom" and add your search lists.



Recommended scan workflow

1) Add QRDI vulnerabilities in debug mode.

- How do I do this? Add **`debug_level:`** to top level JSON object and set logging level to 100, 200, 300 or 400. Learn more

2) Launch scans on QRDI vulnerabilities (Debug mode enabled).

- Scan processing will not be performed for vulnerabilities with Debug mode enabled, and related host based scan data will not be updated in your account.

3) Review scan results and confirm your vulnerability detections are performing as expected.

- You'll notice details for QRDI vulnerabilities include additional sections for debug data and errors (see Scan Results). Use the debug information to analyze the results and resolve any issues you may have.

4) Edit QRDI vulnerabilities and disable debug mode.

5) Launch scans on QRDI vulnerabilities (Debug mode disabled).

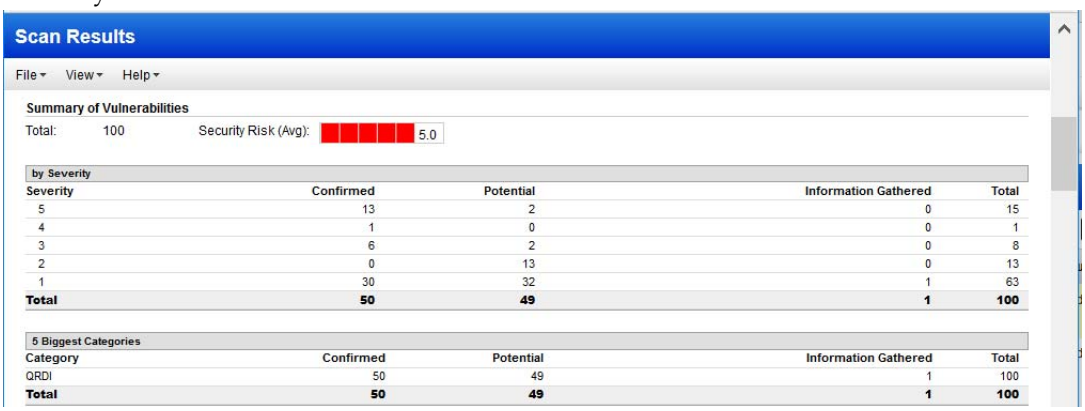
- Scan processing will be performed for vulnerabilities with Debug mode disabled, and related host based scan data will be updated in your account.

Scan Results

Scan results return information on all vulnerability checks performed by the scan.

- All QRDI vulnerabilities, as defined in the option profile, are included in scan results
- Debug info is included in detailed results for QRDI vulnerabilities in debug mode

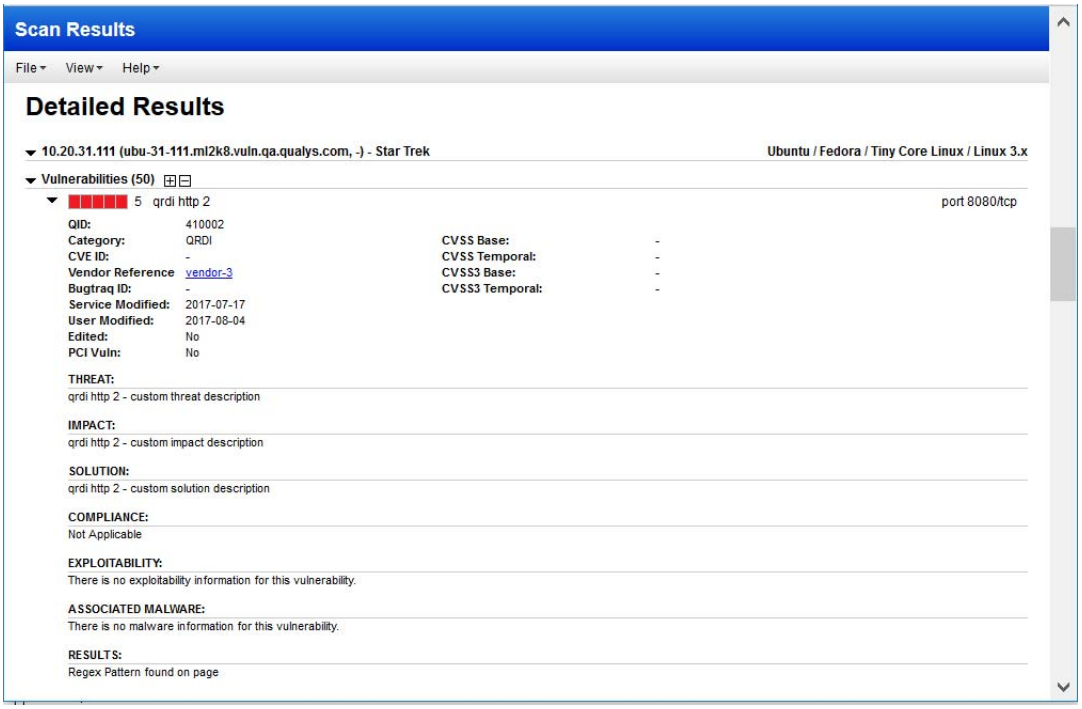
Summary shows vulnerabilities found:.



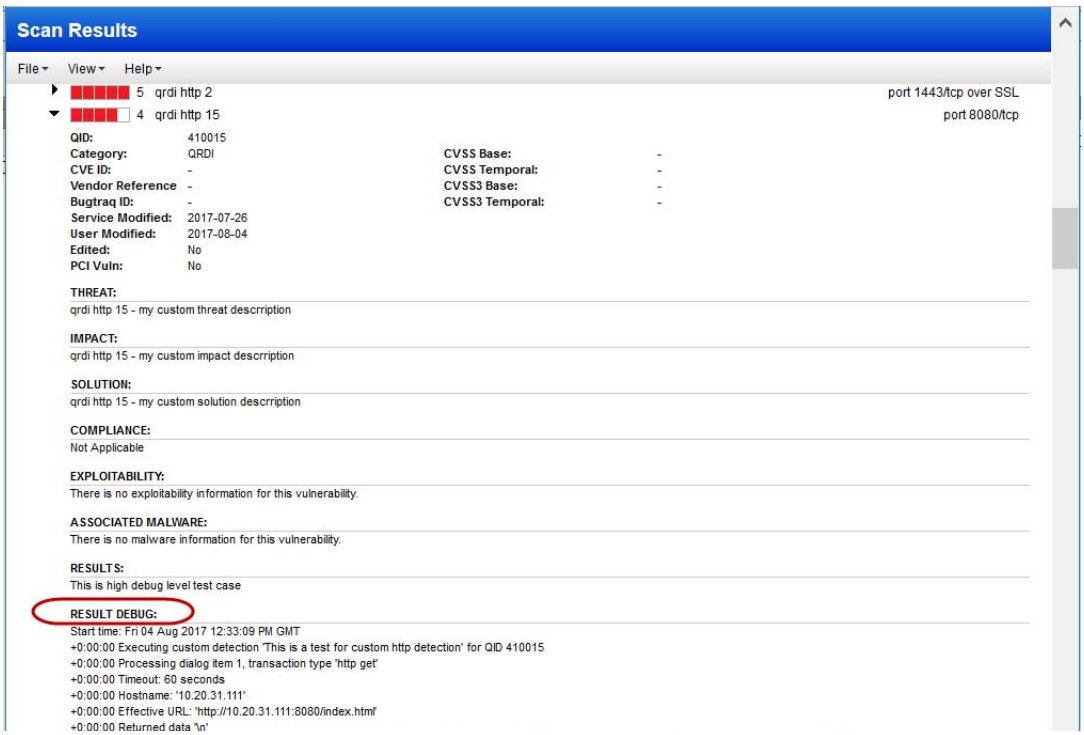
Debug info in scan results

Additional data is passed from the scanning engine and shown in scan results for QRDI vulnerabilities in debug mode. In Detailed Results, RESULT DEBUG contains debug data (log messages), always posted with corresponding QID if trigger condition fulfilled. RESULT ERRORS is present if detection ended in error.

Sample QRDI vulnerability not in debug mode - details like Qualys provided QID:



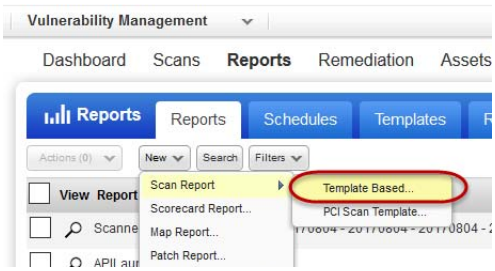
Sample QRDI vulnerability in debug mode - details include section RESULT DEBUG (logging info) and possibly RESULT ERROR if error encountered:



Sample QRDI vulnerability in debug mode with tag <RESULT_DEBUG> (XML format):

```
...
<VULN number="410015" severity="4">
        <TITLE><![CDATA[qrdi http 15]]></TITLE>
        <LAST_UPDATE><![CDATA[2017-08-04T17:20:39Z]]></LAST_UPDATE>
        <CVSS_BASE></CVSS_BASE>
        <CVSS_TEMPORAL>-</CVSS_TEMPORAL>
        <CVSS3_BASE></CVSS3_BASE>
        <CVSS3_TEMPORAL>-</CVSS3_TEMPORAL>
        <PCI_FLAG>0</PCI_FLAG>
        <DIAGNOSIS><![CDATA[qrdi http 15 - my custom threat
descrription]]></DIAGNOSIS>
        <CONSEQUENCE><![CDATA[qrdi http 15 - my custom impact
descrription]]></CONSEQUENCE>
        <SOLUTION><![CDATA[qrdi http 15 - my custom solution
descrription]]></SOLUTION>
        <RESULT><![CDATA[This is high debug level test case]]></RESULT>
        <RESULT_DEBUG><![CDATA[Start time: Fri 04 Aug 2017 12:33:09 PM
GMT
+0:00:00 Executing custom detection 'This is a test for custom http
detection' for QID 410015
+0:00:00 Processing dialog item 1, transaction type 'http get'
+0:00:00 Timeout: 60 seconds
+0:00:00 Hostname: '10.20.31.111'
+0:00:00 Effective URL: 'http://10.20.31.111:8080/index.html'
+0:00:00 Returned data '\n'
+0:00:00 Returned data '<!DOCTYPE html PUBLIC "-//W3C//DTD XHTML 1.0
Transitional//EN" "http://www.w3.org/TR/xhtml1/DTD/xhtml1-
transitional.dtd">\n'
...
+0:00:00 Dialog item 3 finished with action 'stop'
+0:00:00 Custom detection returned success]]></RESULT_DEBUG>
</VULN>
...
```

Reporting



Latest QRDI vulnerability detections appear in reports and API output unless the QRDI vulnerabilities are in Debug mode

QRDI Vulnerability Definition

A QRDI definition, in valid JSON format, describe the logic of the vulnerability detection. This section is the official reference of the JSON schema supported by QRDI.

QRDI detections are based on these third-party technologies:

- The JSON document standard (www. json.org) to formally describe structured data
- The Lua programming language (www.lua.org), currently version Lua 5.3
- PCRE - Perl Compatible Regular Expressions (www.pcre.org)

JSON document overview

The JSON standard supported by QRDI is compatible with ECMA-404. In addition, QRDI permits the use of JavaScript-style comments which start with "/ /" and extend to the end of the line. The top-level item of any JSON document supported by QRDI is a JSON “object”, with some mandatory and some optional fields. HTTP detection type is supported at this time.

The following is an example of a JSON document describing a valid HTTP detection:

```
{
   "detection_type": "http_dialog", "api_version": 1,
   "trigger_type": "service", // may also be "virtual host"
   "title": "custom XSS detection",
   "dialog": [
      {
         "transaction": "http get",
         "object":
"/cgi-bin/no5_such3_file7.pl?\"><script>alert(73541);</script>",
      } ,
      {
         "transaction": "process",
         "mode": "regexp",
         "match":  "\"><script>alert\\(73541\\);</script>"
      } ,
      {
         "transaction": "report", "result": "XSS found"
      }

      }
```

Common concepts

Variables

QRDI supports the notion of “variables”. A variable is a storage slot used to store temporary data during the execution of a detection, e.g. a variable might be used to store part of the result of one query which can then be used in the formatting of a second query, as part of the same detection, or to store data in preparation for reporting. The scope of each variable is a single QRDI detection, i.e. values of variables are not preserved across different QRDI detections.

There are two types of variables: user variables and system variables. System variables have read-only permissions from the user's point of view. They are set to a defined value by the QRDI run-time, and can be read by the JSON detection, but cannot be modified.

Example: With HTTP detections, after every “http get” or “http put” transaction the HTTP body of the result returned from the web server is available in the system variable “body”.

User variables may be freely created, read and written from a JSON detection.

Eval-expressions

Several places in the QRDI JSON schema allow values to be dynamically calculated by using an “eval-expression”. Currently eval-expressions allow the use of constants and string concatenation. The following items are permitted anywhere eval-expressions are allowed:

String constant, e.g.
- **`"abc"`**

Numeric constant, e.g.
- **`123`**
- **`45.6`**

User variable reference, e.g.
- **`{"user": quoted_variablename}`**

System variable reference, e.g.
- **`{"system": quoted_variablename}`**

String concatenation of items in an array, e.g.
- **`{ "concat": [eval-expression  ...]}`**

A more complex concatenation example:

```
{"concat": "abc", {"system": "body"}, 123, {"user": "myvarl"}]}
```

The result of this eval-expression is a string constructed by concatenating the following items:
- "abc"
- The value of the system variable body
- "123" (automatically converted from an integer to a string)
- The value of the user variable myvarl

Dialogs and transactions

Every JSON document describing a QRDI detection has to contain a "dialog" field that describes the dialog underlying the detection, as shown in the example above. The value of a dialog field is a JSON array consisting of one or more transactions. Each "transaction" describes one step of the QRDI detection. Transactions are usually executed serially, in the order specified, but that behavior can be modified by certain "return actions", as described below. Each transaction can have the following fields, plus any fields specific to the transaction type:

- **`"transaction"`**: mandatory field that contains the transaction type as a string constant. The supported transaction types depend on the type of detection.
- **`"label"`**: optional field that assigns a label (name) to a transaction. This is used in combination with the "goto" return action described below, to allow the transaction to be the target of a "goto" jump. Labels have to be unique within a QRDI detection.

The only transaction type shared by all types of detections is the "report" transaction, which has the following format:

```
{
"transaction": "report",
"result":  eval-expression
}
```

A **`"report"`** transaction is always the last transaction run in a detection. It calculates the result from its eval-expression and then posts that result as the cleartext result of the QID associated with the detection. Processing of the detection stops after that.

Processing of the transaction array of a detection can end in one of the following ways:

- By reaching a **`"report"`** transaction. This causes the QID to be posted with the cleartext result provided in the **`"result"`** field of the report transaction.
- By reaching the end of the transaction array. This causes the QID not to be posted.
- By receiving a **`"stop"`** return action (described below). This causes the QID not to be posted.
- By encountering an error, or an **`"error"`** return action (described below). This causes the QID to be posted with an error message in the cleartext section of the QID result and in the /INFO/ERROR XML tag of the XML section of the QID result.

Return actions

"Return actions" are used in several places of the JSON schema. A return action defines the result of a transaction and tells the QRDI run-time what to do next, e.g. to abort the detection, continue with the next transaction, continue with a different transaction, or throw an error. The following return actions are currently supported. Simple return actions, which do not take a parameter, can be specified as constant strings. Return actions that require a parameter are specified using a JSON object.

- **`" continue"`** : this is the default and means that the run-time will continue with the next transaction in the transaction array. If the end of the transaction array is reached then the processing of the detection ends, and the QID is not posted.
- **`"stop"`**: this tells the run-time to stop processing the current detection. The QID is not posted in this case.
- **`"report"`**: this tells the run-time to skip to the next transaction, in sequential order, which is of type **`"report"`**. If no such transaction is found then this behavior is equivalent to **`"stop".`**
- **`{"action": "goto", "label": quoted_targetlabel}`**: this tells the run-time to continue with the transaction that has the specified label. If such a transaction does not exist then an error is thrown, and the detection processing ends with an error condition.
- **`{"action": "error", "message": quoted_errormsg}`**: this causes the processing of the detection to stop and for an error to be thrown. This causes the QID to be posted with an error, as described above.

JSON syntax common to all detections

The following fields are supported in the top-level JSON object and are common to all types of remote detections:

- **`"detection_type"`**: required field and describes the type of the QRDI detection. Currently the only supported type is **`"http dialog"`**.
- **`"api_version"`**: required field and describes the QRDI version for which this detection was written. At this time the only supported value is the integer **`1`**.
- **`"trigger_type"`**: required field which indicates the condition or event that should trigger the QRDI detection. For supported values see HTTP dialog detections.
- **`"debug_level"`**: optional field which indicates the verbosity of debug information (logging) being returned for each instance of a QRDI detection. The following values are permitted:
  - **`0`**: no debug information is generated. This is the default.
  - **`100`**: Only the start and end of a QRDI detection are logged
  - **`200`**: In addition to the information from 100, the start and end of each transaction and run-time errors are logged
  - **`300`**: In addition to the information from 200, the values of variables and temporary data is logged. However string values are only logged if they contain valid UTF-8 strings without special characters, and if the string value has fewer than 1024 characters.
  - **`400`**: Similar to 300, but strings which do not contain valid UTF-8 or which are longer than 1024 characters are logged as well, in a combined binary/hex dump if necessary. For strings longer than 8k8 only the first 4k8 and the last 4k8 are logged.
- **`"dialog"`**: required field that contains the transactions which are run for the detection. The format of supported transactions is described in HTTP dialog detections.

- **`"timeout"`**: optional field that contains the overall timeout in milliseconds. This is the maximum total time spent running the detection. The default is 60000 (60 seconds). The maximum permitted value is 180000 (180 seconds).
- **`"os"`**: optional field that contains a PCRE-compatible regular expression that needs to match the scan target's operating system in order for the detection to be run. The type of OS fingerprint that this field is compared against is described in HTTP dialog detections.
- **`"not_os"`**: optional field that contains a PCRE-compatible regular expression that needs to NOT match the scan target's operating system in order for the the detection to be run. The type of OS fingerprint that this field is compared against is described in HTTP dialog detections.
- **`"ports"`**: optional field that restricts the detection to only run on certain ports. The value can be a single integer or a JSON array of integers. Each integer represents a single port number. The default is to not restrict the ports that the detection is run on, i.e. to run the detection on any port for which the trigger condition or event applies.
- **`"title"`**: optional field that contains a title (description) of the detection. If present the value has to be a string constant. The value has no functional impact on the detection and is only used in some log messages.

Output format

If a QID is posted then the output of the QID can contain the following sections:

- cleartext section: For detections that ended with a **`"report"`** transaction this section will contain the value of the **`"result"`** eval-expression calculated in the **`"report"`** transaction. For detections that ended in an error this section will contain the error message.
- **`<RESULT_ERRORS>`** XML tag in scan results: this tag is only present for detections that ended in an error. In that case the tag contains the error message.
- **`<RESULT_DEBUG>`** XML tag in scan results: this tag will contain the debug data (log messages) generated by the detection during its processing, wrapped into CDATA, if the debug_level was greater than zero. This tag is produced regardless of the end status of the detection, i.e. even if the detection ended in a way that would otherwise not have caused the QID to be posted (e.g. with a **`"stop"`** return action). This also means that with a **`"debug_level"`** greater than zero a detection will always post its corresponding QID if the trigger condition was fulfilled, regardless of its end status.

HTTP dialog detections

Qualys supports HTTP type detections using QRDI. For this type of detection the **"detection_type"** in the top-level JSON object has to be set to **"http dialog"**.

An HTTP dialog detection can be triggered in one of the following two ways:

- With **"trigger_type"** set to **"service"**

  If the scanner engine finds an HTTP or HTTPS service running on a TCP port on the target during TCP service discovery then any HTTP dialog detection with this trigger type is executed on that port. The **"Host:"** field of any HTTP request is set to the IP address of the target.

- With **"trigger_type"** set to **"virtual host"**

  With this trigger type the detection will be run later in the VM scan. After HTTP/HTTPS service discovery the scanner engine performs web server fingerprinting and then attempts to validate the customer-configured list of virtual hosts for the current IP address and port. HTTP dialog detections with this trigger type will then be run for each validated virtual host. The **"Host:"** field of any HTTP request is set to the virtual host name. With this trigger type it is possible for an HTTP dialog detection to be run multiple times on a single IP address and port, once for each virtual host. QID instances generated by these detections are reported and tracked separately on the Qualys Cloud Platform.

The operating system used for matching the **"os"** and **"not_os"** fields in the top-level JSON object is the operating system found during TCP fingerprinting, i.e. the operating system reported in QID 45017 with the **"Technique"** column set to **"TCP/IP Fingerprint"**. That fingerprint may not be accurate if the target is behind a firewall.

HTTP dialog detections support four types of transactions: **"http get"**, **"http post"**, **"process"** and **"report"**. **"http get"** and **"http post"** perform an HTTP transaction on the target, i.e. they send an HTTP GET or POST request to the target and wait for the response. All data encoding and decoding is done by the run-time. **"process"** is used to process the results of the most recent **"http get"** or **"http put"** transactions. **"report"** processing works as described earlier.

"http get" and "http post" transactions

Supported fields:

- **`"object"`**: *eval-expression*. This is a required field that evaluates to the part of the URL that follows the host name and the " : ". It usually starts with a "/" and contains the path, file name and, for GET requests, any encoded CGI values, as required by the target web application. No URL encoding is performed by the run-time, i.e. the value in this field already needs to be in URL-encoded form.
- **`"data"`**: *eval-expression*. This field is required for **`"http post"`** transactions and ignored for **`"http get"`** transactions. It evaluates to the data sent in the body of the HTTP POST request.
- **`"on_error"`**: *return action*. This optional field changes the return action in the case of a network error. The default return action is **`"error"`** with the **`"message"`** field set to a meaningful value, which results in the detection processing to stop in case of an error. If an error encountered during a particular transaction should not result in detection processing to stop then the return action could, e.g., be set to **`"continue"`** or **`"goto"`** with an appropriate target label. Only network errors are affected by this field. HTTP error codes returned by the web server do not trigger this return action and can be handled separately, using **`"http_ status_map"`**, described below.
- **`"http_status_map"`** : By default only network errors result in a return action of **`"error"`**. HTTP transactions which are successful at the network level but result in an HTTP status code different from 200 ("OK") are still considered successful and result in a return action of **`"continue"`**. This optional field allows the return action to be customized based on the HTTP status code. This can, e.g., be used to treat certain types of HTTP status codes (e.g. 404) as errors. The value of this field is a JSON array, and each member of the array is a JSON object with two fields: **`"status"`** and **`"action"`**. The **`"status"`** field can contain a single status code or an array of status codes. The **`"action"`** field contains the corresponding return action. Status codes can be integers, for an exact match, or strings, for a PCRE regular expression match.

  Example:

  ```
  "http_status_map":   [
      {"status": [403,404], "action": {"action": "error", "message":
  "http error"}},
     {"status": ""3[O .. 9] [O .. 9] ", "action": {"action": "goto",
  "label": "process_3xx"}}
  ```

  With this example HTTP status codes 403 and 404 are treated as errors, causing the detection to stop, and HTTP status codes 3xx cause processing to continue with the transaction labeled **`"process_3xx"`**.
- **`"timeout"`** : Optional field that contains the timeout in milliseconds for the HTTP data transfer. The default is 60000 (60 seconds) and only applies the to the HTTP data transfer, not the TCP connection establishment, which has a different fixed timeout of 15 seconds. In addition the overall HTTP transfer is subject to the global time window defined in the top-level JSON object

After the transaction completes, the following system variables are set by the run-time, for use by subsequent transactions.

- **`"body"`**: Content of the HTTP response body (payload).
- **`"network_status"`**: A short text string describing the type of network error, or "OK" if no error has occurred.
- **`"http_status"`**: An integer with the HTTP status code of the response (0 in the case of a network error).

"process" transactions

Supported fields:

- **`"mode"`**: Optional field, which can take one of the following values. The default is **`"substring"`**. This field determines the type of pattern matching that should be performed on the input.
  - **`"substring"`**: the search expression has to be a case-sensitive substring of the source.
  - **`"luapattern"`** : the search expression is interpreted as a Lua pattern, as described in the Lua documentation in the chapter labeled "Patterns".
  - **`"regexp"`**: the search expression is interpreted as a PCRE-compatible regular expression, executed with the PCRE options DOTALL and UTF8.
- **`"match"`**: *eval-expression*. This is a required field which provides the search expression as a string. The exact meaning depends on the **`"mode"`**, as described above. In regexp terms this is called the "needle".
- **`"source"`**: *eval-expression*. This is an optional field which provides the data to search. The default is the content of the system variable **`"body"`**. In regexp terms this is called the "haystack".
- **`"extract"`**: For transactions with a **`"mode"`** of **`"luapattern"`** or **`"regexp"`** it is possible to extract sections from the matched source string and copy them into user variables. This is indicated in the match string by surrounding the corresponding sub-pattern in " () ", as described in the Lua and PCRE documentations. The value of the **`"extract"`** field is a JSON array that contains one JSON object for each variable to be extracted. Each object needs to have the format **`{"var": quoted_varname}`**.

  Variables are mapped to sub-patterns and extracted in the order in which they appear in the array. The first item in the array always matches the complete matched string (index 0 in PCRE). All subsequent items in the array match actual sub-patterns.

  Example:

```
"mode": "regexp",
"source": "abcdefghijkl",
"match":  "b(c.*f)g(h.*j)",
"extract":  [{"var": "v1"}, {"var": "v2"}, {"var": "v3"}]
```

This causes the following user variable assignments:

```
"v1 = "bcdefghij"
"v2 = "cdef"
"v3 = "hij"
```

- **`"on_found"`**: *return action*. This optional field changes the return action in the case that the pattern has matched. The default return action is **`"continue"`**.
- **`"on_missing"`**: *return action*. This optional field changes the return action in the case that the pattern has not matched. The default return action is **`"stop"`**.

Qualys API support

Qualys APIs support QRDI vulnerabilities like Qualys provided ones. We've made no changes to DTDs for XML output. Here's some sample API calls you can make to manage QRDI vulnerabilities in your account.

API v2 support | API v1 support

API v2 support

API v2 calls support scanning and reporting on QRDI vulnerabilities. Several examples provided below.

Launch scan on QRDI vulnerability checks

Launch scan using Scan API v2. The option profile "Initial Options" is configured with all vulnerability checks in the user's account including QRDI vulnerabilities and Qualys provided ones.

API request:

```
curl -u "USERNAME:PASSWORD" -H "X-Requested-With: Curl" -X "POST" -d
"action=launch&exclude_ip_per_scan=64.39.96.0-64.39.111.255&ip=10.10.24.
35,10.10.25.71,10.10.25.80,10.10.25.82,10.10.25.163,10.10.25.165,10.10.2
5.238,10.10.26.98&scan_title=VM_SCAN_API_1500217432&option_title=Initial
Options" "https://qualysapi.qualys.com/api/2.0/fo/scan/"
```

XML output:

See Scan Results (XML format)

Launch host based asset report

Launch asset data report (i.e. host based report) using Report API v2. In sample below the IP 10.20.31.111 has QRDI vulnerabilities detected on it and the sample report shows detection instance (VULN_INFO tag).

API request:

```
curl -u "USERNAME:PASSWORD" -H "X-Requested-With: Curl" -X "POST" -d
"action=launch&report_type=Scan&output_format=html&template_id=4206744&r
eport_title=APILaunchReport_SCAN&ips=10.20.31.111"
"https://qualysapi.qualys.com/api/2.0/fo/report/"
```

XML output:

Simple return as shown below:

```
<?xml version="1.0" encoding="UTF-8" ?>
<!DOCTYPE GENERIC SYSTEM
"https://qualysapi.qualys.com/api/2.0/simple_return.dtd">
<SIMPLE_RETURN>
  <RESPONSE>
    <DATETIME>2017-08-02T21:45:23Z</DATETIME>
    <TEXT>New report launched</TEXT>
    <ITEM_LIST>
      <ITEM>
        <KEY>ID</KEY>
        <VALUE>1665</VALUE>
      </ITEM>
    </ITEM_LIST>
  </RESPONSE>
</SIMPLE_RETURN>
```

Sample report:

```
<?xml version="1.0" encoding="UTF-8" ?>

<!DOCTYPE ASSET_DATA_REPORT SYSTEM
"https://qualysguard.qualys.com/asset_data_report.dtd">
<ASSET_DATA_REPORT>
  <HEADER>
    <COMPANY><![CDATA[Acme]]></COMPANY>
    <USERNAME>acme_rm</USERNAME>
    <GENERATION_DATETIME>2017-08-04T09:54:22Z</GENERATION_DATETIME>
    <TEMPLATE><![CDATA[Host-based-Report007]]></TEMPLATE>
    <TARGET>
      <USER_ASSET_GROUPS>
        <ASSET_GROUP_TITLE><![CDATA[QRDI-AG]]></ASSET_GROUP_TITLE>
      </USER_ASSET_GROUPS>
      <COMBINED_IP_LIST>
        <RANGE network_id="2002">
          <START>10.20.31.111</START>
          <END>10.20.31.111</END>
        </RANGE>
      </COMBINED_IP_LIST>
...
  <HOST_LIST>
    <HOST>
      <IP network_id="2002">10.20.31.111</IP>
      <TRACKING_METHOD>IP</TRACKING_METHOD>
...
      <VULN_INFO_LIST>
        <VULN_INFO>
```

```
            <QID id="qid_410002">410002</QID>
            <TYPE>Vuln</TYPE>
            <PORT>1443</PORT>
            <PROTOCOL>tcp</PROTOCOL>
            <SSL>true</SSL>
            <RESULT><![CDATA[Regex Pattern found on page]]></RESULT>
            <FIRST_FOUND>2017-08-01T09:26:55Z</FIRST_FOUND>
            <LAST_FOUND>2017-08-03T09:05:08Z</LAST_FOUND>
            <TIMES_FOUND>4</TIMES_FOUND>
            <VULN_STATUS>Active</VULN_STATUS>
            <CVSS_FINAL>-</CVSS_FINAL>
            <CVSS3_FINAL>-</CVSS3_FINAL>
            <TICKET_NUMBER>3991</TICKET_NUMBER>
            <TICKET_STATE>OPEN</TICKET_STATE>
          </VULN_INFO>
...
```

Launch scan based asset report

Launch scan based report using Report API v2. In this sample, scan reference `scan/1500204665.47146` performed QRDI vulnerability checks. QRDI vulnerability detections, if any, will be included in the downloaded report.

API request:

```
curl -u "USERNAME:PASSWORD" -H "X-Requested-With: Curl" -X "POST" -d
"action=launch&report_type=Scan&output_format=xml&template_id=4209732&re
port_title=APILaunchReport_SCAN&report_refs=scan/1500204665.47146&ip_res
triction=10.10.25.169&use_tags=1&tag_include_selector=all&tag_set_by=nam
e&tag_set_include=AG_IP_Multi_SA_1486742574"
"https://qualysapi.qualys.com/api/2.0/fo/report/"
```

XML output:

See Simple Return

Sample report:

See Scan Results (XML format)

Download asset search report

Download asset search report for assets with QRDI vulnerability detections using Asset Search API v2. This sample report shows QRDI vulnerability 410002 was detected on the asset with IP address 10.20.31.111.

API request:

```
curl -u "USERNAME:PASSWORD" -H "X-Requested-With: Curl" -X "POST" -d
"action=search&output_format=xml&ips=10.20.31.111&qids=410002"
"https://qualysapi.qualys.com/api/2.0/fo/report/asset/"
```

XML output:

```
<?xml version="1.0" encoding="UTF-8" ?>
<!DOCTYPE ASSET_SEARCH_REPORT SYSTEM
"https://qualysguard.qualys.com/asset_search_report_v2.dtd">

<ASSET_SEARCH_REPORT>
<HEADER>
  <COMPANY><![CDATA[My Corp]]></COMPANY>
  <USERNAME>corp_ab1</USERNAME>
  <GENERATION_DATETIME>2017-08-08T06:25:52Z</GENERATION_DATETIME>
  <TOTAL>1</TOTAL>
  <FILTERS>
    <IP_LIST>
      <RANGE>
        <START>10.20.31.111</START>
        <END>10.20.31.111</END>
      </RANGE>
    </IP_LIST>
    <FILTER_QID><![CDATA[410002]]></FILTER_QID>
  </FILTERS>
</HEADER>
<HOST_LIST>
  <HOST>
    <IP><![CDATA[10.20.31.111]]></IP>
    <TRACKING_METHOD>IP address</TRACKING_METHOD>
    <DNS><![CDATA[ubu-31-111.ml2k8.abc.mycorp.com]]></DNS>
    <OPERATING_SYSTEM><![CDATA[Ubuntu / Fedora / Tiny Core Linux / Linux
3.x]]></OPERATING_SYSTEM>
    <QID_LIST>
      <QID>
        <ID><![CDATA[410002]]></ID>
      </QID>
    </QID_LIST>
    <NETWORK><![CDATA[My Custom Network]]></NETWORK>
    <LAST_SCAN_DATE>2017-08-07T10:00:38Z</LAST_SCAN_DATE>
    <FIRST_FOUND_DATE>2017-08-01T09:30:24Z</FIRST_FOUND_DATE>
  </HOST>
</HOST_LIST>
</ASSET_SEARCH_REPORT>
```

Download Knowledgebase QIDs

Download QRDI vulnerabilities from the KnowledgeBase using KnowledgeBase APIv2. This sample shows details for the QRDI vulnerability with QID 410010.

API request:

```
curl -u "USERNAME:PASSWORD" -H "X-Requested-With: Curl" -X "POST" -d
"action=list&details=All&show_disabled_flag=1&show_vuln_detection_info=1
&ids=410010"
"https://qualysapi.qualys.com/api/2.0/fo/knowledge_base/vuln/"
```

XML output:

```
<?xml version="1.0" encoding="UTF-8" ?>
<!DOCTYPE KNOWLEDGE_BASE_VULN_LIST_OUTPUT SYSTEM
"https://qualysapi.qualys.com/api/2.0/fo/knowledge_base/vuln/knowledge_b
ase_vuln_list_output.dtd">
<KNOWLEDGE_BASE_VULN_LIST_OUTPUT>
  <RESPONSE>
    <DATETIME>2017-08-08T06:26:29Z</DATETIME>
    <VULN_LIST>
      <VULN>
        <QID>410010</QID>
        <VULN_TYPE>Information Gathered</VULN_TYPE>
        <SEVERITY_LEVEL>1</SEVERITY_LEVEL>
        <TITLE><![CDATA[qrdi http 10]]></TITLE>
        <CATEGORY>QRDI</CATEGORY>
        <LAST_CUSTOMIZATION>
          <DATETIME>2017-07-26T09:14:58Z</DATETIME>
          <USER_LOGIN>corp_ab1</USER_LOGIN>
        </LAST_CUSTOMIZATION>
        <LAST_SERVICE_MODIFICATION_DATETIME>2017-07-
21T09:15:09Z</LAST_SERVICE_MODIFICATION_DATETIME>
        <PUBLISHED_DATETIME>2017-07-21T09:15:09Z</PUBLISHED_DATETIME>
        <PATCHABLE>0</PATCHABLE>
        <VENDOR_REFERENCE_LIST>
          <VENDOR_REFERENCE>
            <ID><![CDATA[ven<img src=z>dor-2]]></ID>
            <URL><![CDATA[https://vendor.com]]></URL>
          </VENDOR_REFERENCE>
        </VENDOR_REFERENCE_LIST>
        <CVE_LIST>
          <CVE>
            <ID><![CDATA[CVE-2017-1111]]></ID>
            <URL><![CDATA[http://cve.mitre.org/cgi-
bin/cvename.cgi?name=CVE-2017-1111]]></URL>
          </CVE>
```

```
        </CVE_LIST>
        <DIAGNOSIS><![CDATA[custom threat description]]></DIAGNOSIS>
        <CONSEQUENCE><![CDATA[custom impact description]]></CONSEQUENCE>
        <SOLUTION><![CDATA[custom solution description]]></SOLUTION>
        <PCI_FLAG>0</PCI_FLAG>
        <DISCOVERY>
          <REMOTE>0</REMOTE>
        </DISCOVERY>
        <IS_DISABLED>0</IS_DISABLED>
      </VULN>
    </VULN_LIST>
  </RESPONSE>
</KNOWLEDGE_BASE_VULN_LIST_OUTPUT
```

Create dynamic search list

Create a dynamic search list for all your QRDI vulnerabilities (i.e. **`categories=QRDI`**) using Search List API.

API request:

```
curl -u "USERNAME:PASSWORD" -H "X-Requested-With: Curl" -X "POST" -d
"action=create&title=QRDI-Search_List&global=0&comments=Search List
Created BY API&categories=QRDI"
"https://qualysapi.qualys.com/api/2.0/fo/qid/search_list/dynamic/"
```

XML output:

```
<?xml version="1.0" encoding="UTF-8" ?>
<!DOCTYPE SIMPLE_RETURN SYSTEM
"https://qualysapi.qualys.com/api/2.0/simple_return.dtd">
<SIMPLE_RETURN>
  <RESPONSE>
    <DATETIME>2017-08-08T06:31:46Z</DATETIME>
    <TEXT>New search list created successfully</TEXT>
    <ITEM_LIST>
      <ITEM>
        <KEY>ID</KEY>
        <VALUE>119176</VALUE>
      </ITEM>
    </ITEM_LIST>
  </RESPONSE>
</SIMPLE_RETURN>
```

Download host VM detection data

Using Host VM Detection API v2, VM detection records will list QRDI vulnerabilities for selected assets. No debug info, i.e. logging data or errors, will be shown.

API request:

```
curl -u "USERNAME:PASSWORD" -H "X-Requested-With: Curl" -X "POST" -d
"action=list&ips=10.20.31.111&show_igs=1&qids=410010"
"https://qualysapi.qualys.com/api/2.0/fo/asset/host/vm/detection/"
```

XML output:

```
<?xml version="1.0" encoding="UTF-8" ?>
<!DOCTYPE HOST_LIST_VM_DETECTION_OUTPUT SYSTEM
"https://qualysapi.qualys.com/api/2.0/fo/asset/host/vm/detection/host_li
st_vm_detection_output.dtd">
<HOST_LIST_VM_DETECTION_OUTPUT>
  <RESPONSE>
    <DATETIME>2017-08-08T06:47:52Z</DATETIME>
    <HOST_LIST>
      <HOST>
        <ID>342802</ID>
        <IP>10.20.31.111</IP>
        <TRACKING_METHOD>IP</TRACKING_METHOD>
        <NETWORK_ID>2002</NETWORK_ID>
        <OS><![CDATA[Ubuntu / Fedora / Tiny Core Linux / Linux
3.x]]></OS>
        <DNS><![CDATA[ubu-31-111.ml2k8.abc.corp.com]]></DNS>
        <LAST_SCAN_DATETIME>2017-08-07T10:02:01Z</LAST_SCAN_DATETIME>
        <LAST_VM_SCANNED_DATE>2017-08-
07T10:00:38Z</LAST_VM_SCANNED_DATE>
        <LAST_VM_SCANNED_DURATION>487</LAST_VM_SCANNED_DURATION>
        <DETECTION_LIST>
          <DETECTION>
            <QID>410010</QID>
            <TYPE>Info</TYPE>
            <PORT>8080</PORT>
            <PROTOCOL>tcp</PROTOCOL>
            <RESULTS><![CDATA[Regex Pattern found on page]]></RESULTS>
            <LAST_PROCESSED_DATETIME>2017-08-
07T10:02:01Z</LAST_PROCESSED_DATETIME>
          </DETECTION>
        </DETECTION_LIST>
      </HOST>
    </HOST_LIST>
  </RESPONSE>
</HOST_LIST_VM_DETECTION_OUTPUT>
```

API v1 support

API v1 calls support scanning and reporting on QRDI vulnerabilities. Several examples provided below.

Launch scan v1 (/msp/scan.php)

- Launches scan that performs QRDI vulnerability checks when defined in scan's option profile.

Configure scheduled scan v1 (/msp/scheduled_scans.php)

- Defines scheduled scan that performs QRDI vulnerability checks when defined in scheduled scan's option profile.

Download scan report v1 (/msp/scan_report.php)

- Downloads scan report containing QRDI vulnerabilities when detected by the scan.

Download asset search report v1 (/msp/asset_search.php)

- Downloads asset search report containing QRDI vulnerabilities when detected on target assets.

Download ticket list v1 (/msp/ticket_list.php)

- Downloads remediation ticket list containing QRDI vulnerabilities when detected on target assets.

Download KnowledgeBase list v1 (/msp/knowledgebase_download.php)

- Downloads complete list of QIDs in the KnowledgeBase including QRDI vulnerabilities when added to your subscription.