Background

There is a long-standing bug (BZ438142) reporting Cumin vulnerabilities to web injection attacks. Recently, we decided it was time to take this on. A thorough review of all screens in Cumin uncovered many, many places where the UI is vulnerable to injection of javascript in the displays.

This is a description of where the vulnerabilities were found, how they were fixed, and how Cumin may be tested. It is meant for developers, QE, and members of the Security team.

Misc notes on looking for flaws, reviewing code, and testing

In this discussion, "xml characters" means '&', '<', and '>' as defined by xml.sax.saxutils.escape at http://docs.python.org/library/xml.sax.utils.html. Escaping and xml_escape() refer to a wrapper routine around xml.sax.saxutils.escape.

Whenever a script or a problematic value is called for, this will do: <script>alert(1)</script>

Any line numbers given are against revision 5251

When demonstrating vulnerabilities, note that any of the preventative code can be commented out in revisions 5238 and higher, or alternatively a revision before 5238 can be used.

It also can be helpful when reviewing code to install epydoc and generate the html class hierarchy diagrams, to see where classes are derived from and methods overloaded or attributes set in the __init__ routines.

One testing technique is to run Cumin for a while to gather data, then shut it down and only run cumin-web against the Postgresql database (which will be static at this point). Fields in the database records may be modified from the psql prompt to simulate the creation of objects with fields that contain script values. This may be the quickest/best way to verify that Cumin is escaping values adequately. Note, this doesn't work for values that are gathered through RPC calls from cumin-web (quotas, limits, and Wallaby items).

How problems were neutralized:

There is a single routine in wooly/python/wooly/util.py called xml_escape(value) which is the core routine for nullifying javascript strings. This is a simple wrapper around xml.sax.saxutils.escape() which checks the type of value and calls escape() if the type is 'str' or 'unicode'. The strategy is to call xml_escape() on any value that is displayed on screen and that originates outside of Cumin or through a form submitted by a user. Hard-coded labels in the Cumin source itself are not escaped – if a user can modify the Cumin source, all hope is lost already.

There are a lot of places in Cumin where we could restrict the character sets used to create objects (especially under the Messaging tab). This type of enforcement is done for Submission creation, actually, but it was more of an experiment. The truth is that since virtually all objects visible in Cumin can be created by alternate means outside of Cumin, there is almost no point in restricting character sets on creation. Likewise, since applications outside of Cumin do not necessarily only have Cumin as a consumer, requesting them to restrict their character sets is not optimal (and probably not timely anyway). And of course, we can't count on values from external agents retaining current type and

value characteristics – they could always change. So, we simply fix the displays.

Overview of Cumin vulnerabilities to web injection attacks and associated code:

Here is a list of the types of widgets or pages where vulnerabilities were found, with references to code where appropriate.

Tables:

Tables of values make up a large percentage of the displays in Cumin (75 – 80% or more?). Escaping table cell values and cell titles (optional text displayed for a cell on mouse hover) is relatively simple and provides a lot of coverage from a single point.

The general code to escape table values is in wooly/python/wooly/table.py in the TableColumn class. The do_render_cell_content() and do_render_cell_title() methods dispatch to derived classes to generate content and then apply xml_escape() to the result if self.do_escape is True. The default for self.do_escape is set to True in TableColumn.__init__(), so all tables will have cells escaped unless the value is explicitly changed.

There are two places where general escaping of table cells is turned off (self.do_escape = False) and columns are handled explicitly at a lower level: cumin/python/cumin/grid/quota.py, and cumin/python/cumin/grid/limit.py. These pages can be found at Administrator->Grid->Quotas and Administrator->Grid->Limits. In several of these columns, xml is generated directly for the cell and so escaping (if any) cannot be applied at a higher level.

Breadcrumbs:

This naturally follows from table values. When a table cell contains a link that allows drilling down to a more detailed page, the content of the cell is used to produce a breadcrumb trail of links at the top of the frame for navigation. If table values contain javascript, then the breadcrumbs are also vulnerable once a link has been followed. This is handled in a single place, ObjectViewContextLink.render_content() in cumin/python/cumin/objectframe.py

Statistics Lists:

These are subsets of statistics fields in QMF objects that are displayed in an overview when an item is selected from a table. For example, Adminstrator->Grid->Schedulers->(pick one, goes to Overview)->General stats. Names and descriptions are taken from the xml schema files and are secure. Values are escaped in the StatSet class in cumin/python/cumin/stat.py and its descendents in the render_item_value() and render_formatted_value() methods, which in turn delegate to CuminStatistic.fmt_value() in cumin/python/cumin/model.py.

Note below, this one is academic at the moment. No case exists at present where string type statistics are displayed in a StatSet (see below in the "Setting up problematic values" section)

Details pages:

Similar to statistics lists, these are lists of all the fields of a QMF object, properties and statistics. Names and descriptions come from xml schema files and are secure. Values are escaped in the

render_attributes() method of the ObjectAttributes class in cumin/python/cumin/objectframe.py.

Configuration pages:

Cumin has an interface to the remote configuration system (aka Wallaby) for condor. This interface shows up in part under the Inventory tab and primarily under the Configuration tab. Through Wallaby we can create Nodes (which show up as systems under Inventory), Tags, and Features. Features are assigned to Tags, and Tags are applied to Nodes. In Cumin, we can create Tags and do the assignments (Nodes and Features can only be created outside of Cumin at this point).

Aside from the tables and breadcrumbs discussed above, these values from Wallaby show up in a collection of custom widgets under the Adminstrator->Grid->Configuration tab.

The widgets are all defined in cumin/python/cumin/grid/tags.py.

Messaging UI pages:

The types of displays here are pretty much limited to vanilla tables, details pages, and breadcrumbs as described above and so those things are covered. However, there are quite a few task links leading to forms and most entities created from these forms can be created with problematic names.

There is a lot of code in the messaging UI, primarily located under cumin/python/cumin/messaging. Most of it deals with forms. The best way to experiment here is just to open the forms and start creating/deleting objects of different types.

Cumin user names:

Yes, believe it or not Cumin user names can be javascripts and you can use the script as a login.. When "Hi, X" is displayed at the top of the page it must be escaped. Probably, xml characters should be disallowed in Cumin user names at some point in the very near future.

Setting up problematic values for testing

This section briefly describes how to exercise the problem areas to either 1) demonstrate the vulnerability with an unpatched version of Cumin or 2) verify that the problem is handled in newer Cumin packages (revision 5238 or higher).

Tables:

As noted above under miscellaneous notes, injecting values directly into psql for testing may be fruitful. An example of how to do this is given under the **Details pages** section below. Alternatively, objects may be created the good old fashioned way – here are a few examples.

Submissions with javascript names can be created in a few different ways:

a) Comment out the following check in cumin/python/cumin/grid/submission.py. This will allow creation of submissions from Cumin where the name is a script. Restart cumin-web with this change and submit through the form.

```
# elif xml_escape(des) != des:
#     error = FormError("XML special characters are not allowed in submission descriptions")
#     self.form.errors.add(session, error)
```

b) Submit a job from outside of Cumin. An easy way to do this is to use a modified version of /usr/share/condor/aviary/submit.py, which is installed with the condor-aviary package. On line 76, change the name of the submission to a script -- an example modified script is attached. Note, check the README in /usr/share/condor/aviary and issue the environment command given there before executing submit.py.

```
export PYTHONPATH=`pwd`/module:$PYTHONPATH
```

Submitting a job in either of these two ways will add a submission to the submissions table and will allow testing breadcrumbs by clicking on a submission.

Quotas

The quota displays use custom columns. It is possible to create a quota in condor with a javascript name. Running condor with the attached 80quota.config file placed in /etc/condor/config.d/ will illustrate this problem. Go to Adminstrator->Grid->Quotas for viewing.

Note, there are also exception cases in the Static quota and Dynamic quota columns, when the value will not render to a float. These are hard to test without modifying the code to produce an exception, but xml_escape() calls have been added around these values as well. Paranoia, most likely, but it was tested anyway since the exception value strings may originate outside of Cumin.

Breadcrumbs:

Follow instructions under Tables to create problematic submissions and then select a submission from the submissions table. Or, use psql to manipulate the name of a scheduler or negotiator in the database and drill into that. Breadcrumb code is shared so proving it once is enough.

Statistics lists:

It turns out that there are no statistics in the xml schema files with a string type that are displayed in a statistics overview page. This was tested during development of the fix by ignoring type and injecting values directly from the render calls with modified code.

If someone would really like to test this from the database, the type of a column could be changed to a string type and a javascript set for the column value from psql. But, this is probably overkill. **Details pages** below shows an example of modifying a Postgres value from psql.

Details pages:

There are plenty of statistics and properties on objects that have string types. These can be modified to verify that escaping is taking place on the Details pages. Let Cumin run for a while to populate the database (things will be visible in Cumin), shut the service down, modify the db, and run just cumin-web to display the static data. We shut down the rest of Cumin to make sure that our diabolical changes are not overwritten. In the case below, go to Administrator->Grid->Schedulers->(pick one)-

>Details to view your work.

service cumin start

(let it run for a while)

service cumin stop

Modify a string property on an object in the database, for example Machine on all Schedulers:

$ psql -d cumin -U cumin -h localhost

cumin=# UPDATE "com.redhat.grid"."Scheduler" SET "Machine" = '<script>alert(2)</script>';

$ cumin-web

Configuration pages:

Setting up configuration values takes a bit of extra work when you're not working with a real deployment.

On a monolithic system running condor, Cumin, and the broker, install these packages to allow use of the Wallaby interface in Cumin:

```
# yum install wallaby wallaby-utils condor-wallaby-base-db condor-wallaby-tools
# service wallaby start
```

Changing the wallaby-refresh interval to something like “10” in /etc/cumin/cumin.conf will make this easier to observe because Cumin will update more quickly.

[common]
...
wallaby-refresh: 10

```
# service cumin restart
```

Now create some nodes and features for Cumin to see (some problematic and some not). Nodes in Wallaby roughly correspond to machines, but may be named anything at all.

$ wallaby add-node big_moose
$ wallaby add-node \<script\>alert\(3\)\<\/script\>
$ wallaby add-node \<script\>alert\(4\)\<\/script\>

$ wallaby add-feature antlers
$ wallaby add-feature \<script\>alert\(5\)\<\/script\>
$ wallaby add-feature \<script\>alert\(6\)\<\/script\>

Now go to Adminstrator->Grid->Configuration and click on the “Create tags” link any number of times to create tags (here also, use some problematic names, and some normal ones).

Select a tag and click on the “Edit hosts” and “Edit features” task links to add features to a tag and apply the tag to Nodes. These pages demonstrate all of the custom widgets involved in the Wallaby interface.

In the unpatched version of Cumin, the main displays will show vulnerabilities but so will the dialog pages that allow selection of hosts and features.

Messaging UI:

For the Messaging tab, it's pretty easy to just create problematic objects through the UI. Simply step through the possible pages, clicking on any and all task links and creating or destroying objects as you go along. Alternatively, you can use qpid-config from the qpid-tools package to create objects from the command line, for example

```
# yum install qpid-tools
# qpid-config add queue "<script>alert(7)</script>"
# qpid-config --help
```

Cumin user names:

```
$ cumin-admin add-user "<script>alert(81)</script>"
Enter new password:
Confirm new password:
User '<script>alert(82)</script>' is added
```

Now go to the Cumin login page (log out the current user if you're already logged in).

Log in as user <script>alert(83)</script>. The username will show up at the top of the frame. Also, the username will show up in Submission tables as the owner if Submisisons are made as this user.