Mozilla Security Review

General Information

Project Name:	Datazilla
Security Reviewer:	Matthew Fuller (mfuller), David Chan (dchan)
Date of Review:	6/20 - 6/25
Bugzilla Bug #:	763807

Background Information:

Datazilla (https://github.com/mozilla/datazilla) includes a data model and web service that manages storing and analyzing performance test data for mozilla software products. We will be using it to manage data for a variety of projects including: talos, b2g, stoneridge, jetperf, xperf, and peptest. The intention is to house all performance related test data that the Automation and Tools (ateam) group is responsible for managing.

Unique database instances are used for each project and there is no requirement for co-localization so that projects can scale independently. Datazilla provides the group with a way to re-use data models, web services, and web based user interfaces across multiple projects.

Helpful URLs:

https://github.com/mozilla/datazilla
http://datazilla.readthedocs.org/en/latest/index.html
https://bugzilla.mozilla.org/show_bug.cgi?id=763807

Access Information:

How is the application accessed?
Is the application internal or publicly available?
If it is internal, what mechanism prevents non-members from accessing it?

Datazilla is currently accessible internally only on the MPT VPN at http://10.8.73.32/talos/. ("talos" can be replaced with each instance of the application, i.e. "secreview" was used for this review). Anyone on the MPT network can access the application via the URL.

Eventually, once released, Datazilla will be accessible via a public URL on mozilla.org subdomain or other directory. This URL will be publicly available with no restrictions. It is ingesting data from several VPNs and trusted sources of test data.

Infrastructure and Backends:

What languages do the applications use?
What database language is used if applicable?
Are the running versions up to date?
What server is it running on?

Python, Django, JavaScript, JSON, HTML, CSS, MySQL.

Apache Server (Apache/2.2.15 (Red Hat))

Accounts and Passwords:
If the mechanism to prevent general access is a password, how is the signup process handled?
How is account information stored?
Are passwords properly stored within databases if applicable?
Is a password policy in place?
Are accounts locked-out after a number of invalid logins?
How long are session cookies stored?

There are no user accounts or passwords. The application is simply accessed via a URL available only to MPT network users. Therefore, no user account information is needed or stored.

Oauth is used for clients to POST data to the server and restrict access to clients within the necessary project.

Third-Party Resources:
Are third-party resources used (i.e. JavaScript libraries, images, CSS, etc.)?
Can those resources be trusted / are they from reputable sources?
Is there a chance the resource could be compromised?
Is it possible to host the resources locally to mitigate risks?
Is a third-party responsible for storage of user data?

No, all libraries are hosted locally. No resources are loaded from external sites.

Data Handling:
What kind of data is transferred between the user and the application?
Is this data generated by the user or generated automatically?
Can the data be trusted?
How is the data sent to the application (i.e. JSON format, plain-text, GET/POST, etc.)?
How is the data handled by the server/application?
Can the data be manipulated in transit?
What happens if the data is altered?
What is done with the data once it is received (i.e. stored in a database, displayed to users)?
Is any data storage done via cookies? If so, what kind of data is stored via this method?

Test data from various projects in the form of JSON encoded strings are being sent between the user and the application. Once the application receives the test data from trusted test environments, it is saved in the database, where it is later retrieved and presented to the web page user in the form of charts and graphs. No cookies are used on the web page. The data can be manipulated by the web page viewer by using a proxy to intercept the traffic.

Also, anyone (within MPT) can post data to any project (currently - this is expected to change to only trusted users via oauth or API tokens). If the data is altered, the test data for that particular instance is rendered as invalid. Also, data submitted is eventually reflected back to the user providing a possible attack vector if the submitted data is compromised. However, only trusted sources will be submitting data so this concern is mitigated but must still be considered. The team will be implementing methods to sanitize input and only accepting data that is expected (integers, product name strings, etc.).

[Updated:] Anyone with the correct keys via Oauth will be able to post data to that project alone.

There is no client-side storage for the web application.

For more information on data handling: http://datazilla.readthedocs.org/en/latest/index.html

Data Sensitivity:
What kind of data is being stored and/or manipulated by the application?
Does this data need to be encrypted in transit? In storage?
What is the impact if this data is lost/stolen?

The data is simply test data for various projects and software test instances. The data is not sensitive in nature and does not include personal data, login credentials, or any other form of sensitive information.

Application Administration:
Is there an administration console?
Can it be accessed publicly?
How is it secured if so?
Are correct methods used to prevent admin actions from being performed outside of the admin console (i.e. using CSRF tokens)?

There is no admin console at this time.

Security Coding Flaws:
Have all user inputs been sanitized?
Do all URL variables pass through sanitization?
Is data from databases escaped properly?
Are CSRF tokens used to prevent POSTs from outside websites?
If a database is used, are protections against SQL injection in place?

~~There is currently no sanitization done for the input data (as long as it is a valid JSON string).~~ The URL variables to retrieve data via the API are sanitized and only integers are allowed (test run ID, etc.). ~~Data from the database is not escaped when reflected back to the user via the web front end at the moment.~~ However, only trusted sources are submitting data, although the goal is to escape all data regardless of source. No methods are currently being used to prevent POSTs from unknown sources. The goal is to use API tokens or oauth 1.1.

[Updated:] Data is sanitized via encodeHtmlEntities() or loaded as a hidden input element in the HTML template and subjected to django's autoescape function (see notes for more information)

Testing:
List all tests performed on the application

1. Loaded URL of unknown page
2. Attempted direct API access via http://10.8.73.32/secreview/api/test_runs?_=1340317632288

3. Attempted modification of above URL (2) via proxy with insertion of <script> tags and an alert for XSS testing
4. Loaded URL of invalid test_run number (111)
5. Attempted SQL injection: ‘OR’1’=’1 as test_run value
6. Attempted CSRF via external form
7. Port scan of server
8. Spider scan of site

Results:
List the results of the above tests

1. 500 Server Error
2. Displayed in browser plain text
3. Script tags and alert were escaped and displayed in plain text
4. Showed only result in database (expected, or should this be “unknown”)
5. Showed only result in database (expected, or should this be “unknown”)
6. Submitted data - issue?
7. Open port 80 (web), 22 (ssh), 3306, and 5666 - ssh rejects (Permission denied (publickey,gssapi-keyex,gssapi-with-mic)).
8. Showed /logout, /static, /a -> none loaded in browser (500 errors) and forbidden in static

Meetings and Notes:

6/22 - carljm, jeads, mfuller, rforbes
1. Is the site entirely private (no public access) or will there be public URLs?
2. Is there any way that someone could inject SQL i.e. by including a SQL statement for a test_run number? By creating a fake project name such as ‘OR’1’=’1?
3. Any sort of rate limits to prevent DOS - script could repeatedly submit data
4. Do any libraries connect to remote connections or is everything saved locally?
5. Is it ok that the API return values are accessible directly via URL?

Eventually, we will have a URL that is publicly accessible. The test data will be coming from trusted sources - but they will work on sanitizing it with a whitelist or something similar.

The API URL that loads the data into the set is going to be protected via oauth 1.1 or API tokens.

The ID strings are sanitized via integer only.

No remote library loading.

Ok to return data via URL load - no private data or sensitive data.

EMAILS
Matt,

I wanted to give you an update on what we've done so far to address the security issues you raised before the meeting today. I've included a summary below:

OAuth Info

We have implemented two legged OAuth in datazilla. You can find the OAuth service provider code in:

https://github.com/mozilla/datazilla/blob/master/datazilla/webapp/apps/datazilla/views.py
in the decorator oauth_required

We also implemented a new manage command that implements an OAuth consumer client:
https://github.com/mozilla/datazilla/blob/master/datazilla/webapp/apps/datazilla/management/commands/post_json.py

You can run that code by installing https://github.com/mozilla/datazilla locally and connecting to MPT. The oauth consumer credentials for the secreview project are:

key: [xxxxx]
secret: [xxxxx]

Each project has a unique key/secret associated with it.

Here is an example command line:
python manage.py post_json --project secreview --host s4n4.qa.phx1.mozilla.com --file test.json --key xxxx --secret xxxx

You can run "python manage.py post_json --help" for a full set of options.

The code that generates the oauth consumer key and secret is found in:
https://github.com/mozilla/datazilla/blob/master/datazilla/model/sql/models.py
lines 172-174

We are using the python module http://docs.python.org/library/uuid.html and uuid4 to generate the key and secret and storing the credentials for each project in the datazilla database.

Data Sanitization Info

All data that is displayed in the UI is either passed through the encodeHtmlEntities() function in:
https://github.com/mozilla/datazilla/blob/master/datazilla/webapp/static/js/data_views/Bases.js

or loaded as a hidden input element in the HTML template and subjected to django's autoescape function.

We also made a variety of changes to the function that deserializes the JSON called load_test_data in:
https://github.com/mozilla/datazilla/blob/master/datazilla/model/base.py
to improve error capture.
Jeads

Meeting 7/3 - jeads, carljm, mfuller, dchan
Don’t need Oauth for Talos test data because they’re running older version of Python. Agreed, since biggest risk would be unknown data being submitted, which is being sanitized before input and encoded for output.