Harvester Service v1.0 Testing

5 June 2008

THE AUSTRALIAN NATIONAL UNIVERSITY

Harvester Service v1.0 Testing

Document History

Scott Yeadon	2 June 2008	Version 1.0	Initial document.

Table of Contents

Functionality

Cross Platform Checks
The Harvester Service makes use of Postgres as its underlying database and Tomcat for the servlet container. Both these products are platform-independent with versions available across all major operating systems. The application itself is written using platform-independent Java and while the core of the application is thread-based, the Timer classes are used in order that the system clock of the underlying operating system manages the scheduling of harvests. In addition no priority handling is implemented thus reducing chances of thread prioritisation problems from platform-dependent threading implementations. (This probably wouldn't be a problem anyway given use of the Java Timer classes should work appropriately depending on OS.)

XHTML Validation
The Harvester has no user interface component so does not produce or consume XHTML.

CSS Validation
The Harvester has no user interface component so does not produce or consume CSS.

Accessibility
The Harvester has no user interface component so has no accessibility requirements.

HTTP Service Calls
Services have been tested with missing and invalid parameters to ensure appropriate error responses are generated.

Security

Note that none of these comments apply to tthird party software, only the Harvester software is commented on. Issues in Postgres/Tomcat security and configuration arenot covered here and the documentation of these applications needs to be consulted for relevant information.

Authentication and Authorisation
The Harvester has been developed as a back-end service and currently relies on the front-end client to manage authentication and authorisation. The only authentication/authorisation available is that which is configurable at the server, webserver and servlet container level.

OWASP

The following sections provide comment against each of the OWASP top 10 serious web application vulnerabilities (see http://www.owasp.org/index.php/Top_10_2007).

A1 – Cross-Site Scripting (XSS)

The Harvester has no browser interface and HTTP responses are provided in XML format so XSS is not an issue.

The harvest modules for ORCA and BEST do not check the content of their harvests. In the case of OAI-PMH, an OAI-PMH harvest will fail if the content does not contain XML and does not contain XML elements required for OAI-PMH request processing. As a precaution any service acting as a recipient of POST-ed data from the Harvester should not pass the information to a browser before checking the content. This is especially true for GET method harvests which will simply pass data to a consumer without any checking.

A2 – Injection Flaws

Input Validation. Input validation is performed against HTTP service parameters to ensure they conform to the required format and are valid within the context of the Harvester.
Use strongly typed parameterised query APIs. Strongly typed prepared statements with substitution placeholders are used for all database operations.
Enforce least privilege. Database operations are all performed by a single database user whose accesses can be controlled by the database adminstrator. Access to HTTP services needs to be controlled by a combination of client interfaces and server configuration (e.g. port access, ip filtering at web server/servlet container). BEST and ORCA control access to services based on their authorisation mechanisms.
Avoid detailed error messages. Error messages held within HTTP service responses are detailed enough to inform the user. Some exception responses may provide information useful to an attacker, however these are limited to the exception message and will only be returned in the event of unforseen system failures. Full stack traces are provided only in the Harvester log.
Show care when using stored procedures. Stored procedures are not used in the Harvester, all SQL is contained in a DAO layer containing SQL fragments for use in prepared statements.
Do not use dynamic query interfaces. The Harvester does not provide means for executing dynamic queries by external agents.
Do not use simple escaping functions. The Harvester does not use any escaping functions.

A3 – Malicious File Execution
Users of the Harvester are limited to HTTP services available via some client application. The client application and server admin are responsible for ensuring these services are secure through a combination of application and server security mechanisms.. Where content processed by the Harvester is not XML errors will occur, however any client application should check the data being received to ensure data is in a form the client application will expect. This is especially important if making use of the GET harvest method.

A4 – Insecure Direct Object Reference
Access to harvest records is available using a key such as the harvest ID. It is important that client applications secure access to harvest ID information and the services around individual harvest IDs. The Harvester itself should be configured (via server or servlet container settings) to only allow connection via client application workstations. Clients should ensure the harvest IDs they mint are not guessable.

A5 – Cross Site Request Forgery
The Harvester does not make use of a web browser so cannot initiate these attacks, however is a potential target of such attacks.

A6 – Information Leakage and Improper Error Handling
All exceptions are logged in the Harvester log file, those which cannot be handled are thrown to the servlet container for handling. Error messages in HTTP service responses relate directly to the request although an Exception will result in the message string from the Exception being displayed. The information in the Exception will not provide information outside the consumer’s context, however it is possible that the servlet container may provide stack trace information should some form of internal error occur.

A7 – Broken Authentication and Session Management
Authentication for the Harvester is currently provided for at the server and servlet container level and needs to be configured on a per-installation basis. Beyond this, client applications currently have the responsibility for securing access to particular services.

A8 – Insecure Cryptographic Storage

Not Applicable. The Harvester as it currently stands provides temporary storage of publicly available information. It does not store sensitive information such as user names and/or passwords.

A9 – Insecure Communications
It is recommended the Harvester be configured on a trusted server and only communicate with trusted clients and servers using HTTP over SSL.

A10 – Failure to Restrict URL Access
URL access can currently be restricted in the servlet container/web server or through client applications. The restriction mechanism will depend on the context of deployment.

Load & Performance

Testing was undertaken in order to locate:

- any bottlenecks in the database
- any bottlenecks in the application
- any bottlenecks in services (i.e. external applications)

Test platform specifications
Linux RHEL 4 x64 Virual Machine
1 vCPU (share level - normal)
2048 vRAM (share level - normal)
36GB vHDD (share level - normal)

Tests undertaken

Concurrent requests
Simultaneous submission of 50 harvest requests were processed in under one second. Harvest requests are the most intensive service request.

Concurrent harvests
Submission and execution of increasing numbers of concurrent harvests running on the test platform in default JVM memory (64MB). Harvests were conducted on valid OAI-PMH Data Providers with each ListRecords response returning 100 registry object records per response. Results indicated 20 concurrent harvests were able to be run under these circumstances without risk of memory issues. The possibility of heap space being exhausted increases if further harvests are scheduled without increasing the minimum JVM heap size.

The number of concurrently running harvests will be dependent on web server, OS and whether other applications are running concurrently in the same JVM. Note that the tests were conducted in an environment not reflective of a production service where all applications involved in the test were hosted on the one machine using the same database (also co-located on the test server) and accessing the same web server (also co-located on the test server).

The alternate GET harvest method was not tested in a performance context as it is a known limitation and should only be configured for small XML (<2MB) documents.

No bottlenecks were encountered at the database level and given the simplicity of the database thousands of harvests managed from the one database would be possible. Response fragments are stored in the database for the duration of the harvest and then deleted. There is likely to be a large number of fragment records on a reasonably loaded harvester application, so it is recommended frequent database defragmentation be undertaken.

General Comments

There were no bottlenecks other than memory use encountered in the application, however under a heavily loaded system database/disk and CPU usage could be significant. Increasing the amount of memory will alleviate Java heap space issues and server clustering may be a means to manage large numbers of harvests. No memory leaks were discovered however memory use monitoring was rudimentary simply analysing logged free memory use as reported by the JVM during running of concurrent harvests.

On systems running a large number of concurrent harvests it is possible the maximum number of processes could be hit. Each harvest requires two threads, one to manage the scheduling and status of a harvest and another for the harvest itself. This depends somewhat on the OS and how or whether it differentiates between processes and threads.

The major known bottleneck is the GET harvest. This currently is based around DOM processing meaning that even a few large harvests using this method will blow out heap space. A possible solution to this is to have a SAX-based GET harvest however this was deemed out of scope since it seems reasonable in the environment the Harvester will operate in to expect clients to be using OAI-PMH Data Providers and fragmenting their responses. Further improvements may be made by using SAX event processing rather than DOM however this is more appropriate for socket-level communications (not application-level) and will also increase complexity and reduce flexibility of individual harvest classes.

No service bottlenecks were encountered.