Table of Contents

UVOS Test Plan

Service/Component Description

This test plan covers two main components: UVOS server and UVOS Command Line Client (UVOS CLC). Additionally testing of two extensions of the UVOS server is also included here: UVOS web applications and UVOS web authentication. All components use UVOS common library which is not installed/distributed separately.

UVOS server description is available in UVOS Service Reference Card.

UVOS CLC is a command line utility which allows for invoking all operations offered by the server. UVOS CLC can work in interactive mode or using command line parameters.

UVOS server is typically deployed as a grid or VO central server, serving multiple sites. UVOS CLC is installed on any machine which has access to the server machine (e.g. administrator's computer).

The components' versions corresponding to this plan are:

- UVOS server: 1.4.1
- UVOS client: 1.4.1
- UVOS web-authn: 1.5.0
- UVOS web-application: 1.5.0

Information about extensions testing

UVOS server extensions are currently not tested automatically. This is due to fact that those are web applications, providing mostly WWW UI. To guarantee high quality we plan to:

- Develop an automated functional test for web authentication module. As the module has hardly any Java code inside, such a test should eliminate need for unit tests of this component.
- Prepare a procedure to manually test web registration module. It is possible (but unlikely) that some unit or other automated tests will be provided too. The procedure will be used to manually check if the interface is working correctly for all the important possible usage flows.

Functionality tests

[TO BE DONE - Add remaining cases, the current list is complete only for the initial release in EMI-1.]

Group creation (implemented: uvos_grp_add)

Test adding a group to the root and as a subgroup. Addition of existing group must do nothing. Newly added groups must be listed in getGroupContent() output.

Group copy and move (implemented: uvos_grp_copy)

Test coping and moving a group. Copied/moved groups must be listed in getGroupContent() output. Moved group must disappear from the old parent. Copied group must stay in the original location too.

Remove group (implemented: uvos_grp_del)

Test removing a group. Removed group must not be included in the getGroupContent() output performed on the parent group. The test must be performed on an empty group and on non empty group with recursive argument set to true. Removal of non-empty group without recursion turned on must fail.

Integration tests

TBD - describe tests from uvos-clc

Performance tests (implemented)

Compute operations per second for the following cases. Passing thresholds are provided in brackets - generally higher expectations for the (more commonly used) read operations are given. Tests are assumed to be performed on a decent workstation.

1. IDENTITY CREATION [min 100 ops/s]
2. GROUP CREATION [min 100 ops/s]
3. NESTED GROUP CREATION [min 50 ops/s]
4. ADD TO GROUP [min 100 ops/s]
5. ADD GLOBAL ATTR [min 100 ops/s]
6. ADD GROUP ATTR [min 100 ops/s]
7. ADD ID@GROUP ATTR [min 100 ops/s]
8. UPDATE GLOBAL ATTR [min 100 ops/s]
9. UPDATE GROUP ATTR [min 100 ops/s]
10. UPDATE ID@GROUP ATTR [min 100 ops/s]
11. GET ALL IDENTITIES [min 100 ops/s]
12. GET ALL IDENTITY'S IMPLIED GROUPS [min 200 ops/s]
13. GET ALL IDENTITY'S DIRECT GROUPS [min 200 ops/s]
14. GET ALL IDENTITY'S ATTRIBUTES (true, true, true) [min 200 ops/s]
15. GET ALL IDENTITY'S ATTRIBUTES (true, true, false) [min 200 ops/s]
16. GET ALL IDENTITY'S ATTRIBUTES (true, false, false) [min 200 ops/s]
17. GET IDENTITY'S GLOBAL ATTRIBUTES [min 200 ops/s]
18. GET IDENTITY'S ATTRIBUTES@GROUP (true, true, true) [min 200 ops/s]
19. GET GROUP CONTENTS with at least 10 entries in each group [min 200 ops/s]
20. REMOVE FROM GROUP [min 100 ops/s]
21. NESTED GROUP REMOVAL with groups containing 100 levels-deep structure [min 10 ops/s]
22. IDENTITIY REMOVAL [min 100 ops/s]

Scalability tests (implemented)

For 20 seconds run in parallel loops with the following operations. There should be no error reported. Memory consumption must be monitored: no more then 10MB increase in total memory usage may be observed (this is allowed as server can cache some information).

1. check for membership
2. get all identity's groups
3. get all identity's attributes
4. get all equivalent identities
5. get group contents
6. get all identities
7. add identity to a group; remove it
8. add two hierarchical groups; add a member to the child group; remove the parent group
9. add two scoped attributes; update value of one; remove them
10. add three identities equivalent to existing ones; remove them
11. add two groups; remove them
12. add 10 identities; add them to an existing group; remove them
13. add identity; remove it

Standard Compliance/Conformance tests

TBD

Regression tests and unit tests

Unit tests coverage must be included in the test report. All bugs reported should have an automated regression test attached if it is possible. Otherwise manual bug checking procedure should be added to this section. Note that this applies to bugs reported from the 1.11.2010.

Bugs that will not get a regression test:

- #3020090. Fix was a change in architecture: now security context is not handled manually by implementation but it is stored as a thread-local data.

Regression tests to be performed manually:

- N/A

Deployment tests

1. Install both the server and the client from the binary packages.
2. Create a keystore that will be used as a server's keystore and truststore. Make sure that this keystore file is readable by the user `unicore`.
3. In the file `/etc/unicore/uvos-server/uvosServer.conf` configure the following properties to use the keystore created in the previous step:
    1. uvos.server.keystore
    2. uvos.server.keystoreType
    3. uvos.server.keystorePasswd
    4. uvos.server.keystoreAlias
    5. uvos.server.keyPasswd
    6. uvos.server.https.truststore
    7. uvos.server.https.truststorePasswd
    8. uvos.server.https.truststoreType
4. Initialize UVOS server database using unicore-uvos-server-initdb command (no-arg), run as user `unicore`.
5. Start service using system init script by root.
6. There should be no errors/warnings in the log file.
7. Server's process uid should be 'unicore'.
8. In the file `/etc/unicore/uvos-clc/uvosClient.conf` set the following properties to use the same keystore and settings as the server:
    1. keystore
    2. keystoreType
    3. keystorePasswd
    4. keystoreAlias
    5. keyPasswd
    6. truststore
    7. truststorePasswd
    8. truststoreType
9. A request to list all known identity types must be performed using the command line client. The result should contain three types.
10. There should be no errors/warnings in the server's log file and on client's console.
11. Stop the server.
12. Install extensions using binary packages: uvos-webauth and uvos-webapp.

13. Start service using system init script by root.
14. There should be no errors/warnings in the log file.
15. Perform HTTP GET request on the address: https://localhost:2443/uvos-webapp-COMPONENTVERSION/VOapplications.do. The empty web registration form should be presented.
16. Perform HTTP GET request on the address: https://localhost:2443/uvos-webauth-COMPONENTVERSION/VOauthentication.do. The login form should be presented.

This topic: EMI > UNICOREUVOSSVVP
Topic revision: r8 - 2011-03-17 - unknown