ALLSEEN ALLIANCE

AllJoyn™ Security 2.0

Feature Test Plan

January 21, 2015

Contents

Reference Documents

Reference Number	Document Name	Document Number	Document Path/Location/Link
1	Security 2.0 design		https://wiki.allseenalliance.org/core/security_enhancements

Acronyms

Acronym/term	Description
CI	Continuous Integration
FR	Factory Reset
MC	Membership Certificate
ASC	AllJoyn Standard Client
ATC	AllJoyn Thin Client

1 Introduction

Security 2.0 is an enhancement feature that will allow an application to validate access to secure interfaces or secure objects based on policies installed by the owner. The controlee will have access to Secure Interfaces or Secure Objects based on these policies. In addition to the encrypted messaging (using AES CCM) between the peers, the Security 2.0 Permission management module manages a database of access credentials and Access Control Lists (ACLs). The AllJoyn™ Core Permission Management component will also do enforcement including concept of mutual authorization before any message action can be taken. It will not be up to the user to enforce permission but user can dictate how application will perform based on the access control lists (ACLs) defined for the application.

The end user Security manager is an optional service that helps user with key management and permission rule building. The security Manger is optional because the permissions can be installed directly into the application. The end user Security manager is, as of now out of the scope of this document as it is being developed by other alliance members.

This document outlines the test plan and test execution strategy for the security enhancements for the 15.04 release. Prominent developments/enhancements of this feature include:

- Enhancements to existing ECDHE_DSA mode of authentication to enforce peer trust.
- Claiming of devices
- Policy management
- Access control
- Enhancements to keystore file to store identity certificates, policies, and membership certificates.

The intent of this test plan is to create a standard, mutually agreed upon document, which will enable the test team to make an informed decision about the overall test scope and coverage will be for this product. Once a standard has been established; the success or failure of the features within the scope can be quantitatively measured, and appropriate feedback can be provided the stake holders.

1.1 Test Process

The testing activity will comprise development of detailed test cases and executing them. The functional testing will be conducted on the Linux platform for AllJoyn standard client and AllJoyn thin client libraries. Some end-end-end tests will be conducted across opposite endianness machines to test over-the-wire messaging between buses.

Discrepancies or potential enhancements identified while testing on the feature branch will be reported in AllSeen JIRA.

1.1.1 Test effort estimate and schedule

Table 1

#	**Feature**	**Estimated time**
1.	Writing test cases	4 weeks
2.	Test execution	TBD

1.2 Test Entrance and Exit Criteria

1.2.1 Entrance Criteria

- Feature branch should be signed off as development complete and ready for testing by the development lead for the feature. `
- Test applications should be available for testing. This includes unit tests, end-end-end system test applications.
- CI builds and automated tests (if any) should pass on the feature branch.

1.2.2 Exit Criteria

Testing will conclude after all the test case results have been agreed upon by the stakeholders.

1.2.3 Assumptions

- Security Manager Module is not in the scope of testing this feature.
- After claiming a device, “No Policy” or “empty Policy” means “implicit deny” for all.
- Policy is never checked while sending a message (Method call or Signal).
- To send a message, the sender should have a membership certificate with auth. data to do so.
- Security 2.0 works only with ECDHE_DSA. Other authentication modes like SRP, RSA, etc. are not supported.
- Test plan is subject to change as the HLD of the feature undergoes any change.

2 Test Strategy

The test cases are divided into the following areas:

- Claiming of devices
- Policy management
- Access control
- Manifest
- End-to-End test
- Memory profiling

The test cases (as specified in Section 3) will be run for standard client and thin client. The functional tests are platform and binding independent and hence will be run on Linux platform only for C++ bindings. An end-to-end test will be run on different endianness machines (OpenWRT and Linux). The term “device bus” means that it is a BusAttachment for both ASC and ATC.

3 Test Coverage

A high-level overview of the test cases is presented below.

3.1 Claiming of devices

Test Objective	**Procedure and Verification**	**Applicable for Standard client and Thin client**
Verify the public key of the device before claiming.	Configure a device with “Claimable state”. The device is in FR condition. The device emits a PermissionMgmt.Notification. Verify that claimableState = ‘claimable’, policySerialNum = 0. Store the public key of the device. Another bus calls GetPublicKey() on the device bus. Compare the public key returned by the API with that coming from the notification. They should match.	Yes
Claim a device using ECDHE_NULL mode	Configure a device with "Claimable" state. The device is in FR condition. Claim the device by the admin bus using ECDHE_NULL mode. Check for PermissionMgmt.Notification. Check for new GUID (The new GUID is the GUID set by the admin bus which should be different than the device's keystore file), claimableState='claimed' , policySerialNum=0. Call GetPublicKey() on the claimed device from another bus and verify the public key of the claimed device. Call GetIdentity() on the claimed device by the above bus and verify the Identity certificate. The identiy cert returned by the API should be the same as that generated by the admin bus.	**Yes**
Claim a device using ECDHE_PSK mode	Configure a device with "Claimable" state. The device is in FR condition. Claim the device by the admin bus using ECDHE_PSK mode. Check for PermissionMgmt.Notification. Check for new GUID (The new GUID is the GUID set by the admin bus which should be different than the device's keystore file), claimableState='claimed' , policySerialNum=0. Call GetPublicKey() on the claimed device from another bus and verify the public key of the claimed device. Call GetIdentity() on the claimed device by the above bus and verify the Identity certificate. The identiy cert returned by the API should be the same as that generated by the admin bus.	**Yes**

Test Objective	Procedure and Verification	Applicable for Standard client and Thin client
Claim a device using SRP mode	Configure a device with "Claimable" state. The device is in FR condition. Claim the device by the admin bus using SRP mode. Check for PermissionMgmt.Notification. Check for new GUID (The new GUID is the GUID set by the admin bus which should be different than the device's keystore file), claimableState='claimed' , policySerialNum=0. Call GetPublicKey() on the claimed device from another bus and verify the public key of the claimed device. Call GetIdentity() on the claimed device by the above bus and verify the Identity certificate. The identiy cert returned by the API should be the same as that generated by the admin bus.	**No**
Claim an an already claimed device.	Claim a device using ECDHE_NULL mode. Try to claim it again using the same admin bus. Claim method-call should fail. Error should indicate that the device is already claimed.	**Yes**
Claim an already claimed device by a different non-admin bus.	Try to claim an already claimed device by a different non-admin bus, get an Error. Error should indicate that the device cannot be claimed non-admin.	**Yes**
Claim a “Non-claimable” device.	Configure a device with "NotClaimableState". The device is in FR condition. Claim the device by the admin bus using ECDHE_NULL mode. Claim should get an appropriate error indicating the device cannot be claimed.	**Yes**
Claim and Reset	Configure a device with "Claimable" state. The device is in FR condition. Claim a device. Check the PermissionMgmt.Notification for Check for new GUID (The new GUID is the GUID set by the admin bus which should be different than the device's keystore file), claimableState='claimed' , policySerialNum=0 Reset the device by the above bus. Check the PermissionMgmt.Notification for Check for GUID (The GUID should be same as above), claimableState='Unclaimed' policySerialNum=0 Claim the device again from a new bus. Check the PermissionMgmt.Notification . Check for new GUID (The new GUID is the GUID set by the admin bus which should be different than the device's keystore file), claimableState='claimed' , policySerialNum=0	**Yes**
Reset the device by non-admin bus.	Reset the device by a non-admin bus, get an Error. Error should indicate that the device cannot be reset by non-admin.	**Yes**
Claim a device using an incorrect identity certificate.	Admin generates an identity certificate based out a a different public key. This public key is not the actual public key of the device. Claim the device. Claim should return an error. Error should indicate that an incorrect identity certicate was passed.	**Yes**

Test Objective	Procedure and Verification	Applicable for Standard client and Thin client
Identity certifcate is signed by admin using a different public key.	Admin generates an identity certifcate signed by a different key. The key used for signing is different than the public key of the admin. Claim the device. Claim should return an error. Error should indicate an incorrect identity certifcate.	**Yes**

3.2 Installation of policy, Membership certificates

Test Objective	Procedure and Verification	Applicable for Standard client and Thin client
InstallPolicy is called by a non-admin bus.	InstallPolicy is called by a non-admin bus. It should return error. Error should indicate that the Method can be called by the admin bus only.	**Yes**
InstallPolicy and GetPolicy	Create a policy, note the serial number. call InstallPolicy using the admin bus. MethodCall should return success. Check the PermissionMgmt.Notification.policySerialNumber. It should match with the above serial number. call GetPolicy using a non-admin bus. It should return error. Error to indicate a permission problem. call GetPolicy using an admin bus. Method should be successfull. Compare the policy returned with the policy created. They should match.	**Yes**
RemovePolicy is called by a non-admin bus.	RemovePolicy is called by a non-admin bus. It should return error. Error should indicate that the Method can be called by the admin bus only.	**Yes**
Install a policy without a serial number.	Setup a policy and ensure that the serial number is not set. InstallPolicy is called with the above policy. Method call should fail. The error should indicate invalid args.	**Yes**
Install a policy with the same serial number.	The device bus where the policy is being installed should return an error. This is because if the policy is installed, another Notification will be sent with the same serial number. Error should indicate “Policy with the same serial number already exists.)	**Yes**
Install a policy with the serial number lesser than the previous policy.	InstallPolicy should return an error. A new policy installed should have a serial number greater than the previous.	**Yes**

Test Objective	Procedure and Verification	Applicable for Standard client and Thin client
InstallPolicy, RemovePolicy	Install a policy on a device bus. Check for PermissionMgmt.Notification.policySerialNumber. It should match with the policy serial number. Remove the policy on the device bus. Check for PermissionMgmt.Notification.policySerialNumber. It should be 0. Call GetPolicy on the device bus. It should fetch empty policy set.	**Yes**
Membership certificate is not signed by the admin.	InstallMembership is called by any bus but it uses a membership certificate not signed by the admin or guild authority. Method should fail with Permission denied.	**Yes**
Membership certifcate is non X509 DER format.	InstallMembership should return error at the sender side indicating “Invalid certificate format”	**Yes**
InstallMembership and RemoveMembership	InstallMembership is called by admin bus. The membership certificate is signed by the admin. The MC belongs to GUILD1. Methodcall is successful. InstallMembership is called again. The membership certificate is signed by the admin. The MC belongs to GUILD2. Methodcall is successful. RemoveMembership is called using the serial number related to GUILD2. Methodcall is successful. RemoveMembership is called using the serial number related to GUILD1. Methodcall is successful.	**Yes**
RemoveMembership is called by a non-admin bus.	RemoveMembership is called by a non-admin bus. It should return error (permission denied.)	**Yes**
RemoveMembership called on an invalid serial number.	RemoveMembership called on an invalid serial number. Methodcall should return error. Error indicating that there is no such certificate installed on the device.	**Yes**
InstallMembershipAuthData is called with a serial number not repesenting the membership certificate.	InstallMembershipAuthData is called on the device bus with a serial number not representing the membership certificate. Methodcall should return error. Error indicating “certificate not found”.	**Yes**
InstallMembershipAuthData is called on an existing serial number but for a different membertship certificate.	admin installs Membership certificate 1 with (GUILD1, serial1) admin installs Membership certificate 2 with (GUILD2, serial2) admin calls InstallMembershipAuthData with serial1 but for Membership certificate 2. The method call should return error indicating "Invalid hash digest"	**Yes**

Test Objective	Procedure and Verification	Applicable for Standard client and Thin client
InstallIdentity is called by non-admin bus.	InstallIdentity is called by non-admin bus. Methodcall should return error (permission denied.)	**Yes**
Install a new identity and GetIdentity.	Admin claims a device. Admin calls GetIdentity on the device. Admin matches the issuer of the certficate with itself using GetIdentity. Admin generates an identity certificate for the device and he changes some fields. (The subject field and the issuer filed still remains the same.) Admin bus calls InstallIdentity with the new certificate. InstallIdentity is successful. A new bus calls GetIdentity on the device bus to check that the identity cert of the device has changed.	**Yes**
InstallIdentity is called using a non X.509 DER format.	The method call should return error at the sender side indicating “Invalid certificate format”	**Yes**
PolicyInstallation works only with ECDHE_DSA, does not work with SRP mode.	Admin claims a device bus using SRP mode of authentication. Admin and device bus clear keys of remote peers. Establish a SRP based session between admin and device bus. Admin calls Install policy on device bus. Error should indicate both sides cannot verify trust establishment with each other.	**No**
PolicyInstallation works only with ECDHE_DSA, does not work with RSA mode.	Admin claims a device bus using RSA mode of authentication. Admin and device bus clear keys of remote peers. Establish a RSA based session between admin and device bus. Admin calls Install policy on device bus. Error should indicate both sides cannot verify trust establishment with each other.	**No**
PolicyInstallation works only with ECDHE_DSA. Keystore not cleared.	Admin claims a device bus using SRP mode of authentication. Admin calls Install policy on device bus. (The keys have not expired and thus an SRP based session is established.) Error should indicate both sides cannot verify trust establishment with each other.	**No**
Claim using SRP mode of authentication, but InstallPolicy using ECDHE_DSA mode.	Admin claims a device bus using SRP mode of authentication. Admin and device bus clear keys of remote peers. Establish a ECDHE_DSA based session between admin and device bus. Admin calls Install policy on device bus. This should be successful.	**No**

3.3 Access Control

Test Objective	Procedure and Verification	Applicable for Standard client and Thin client
Test the DENY action mask for a method call, property, signal for ANY_USER	A device bus has DENIED action on a signal, method call and a property for ANY_USER. Another peer bus has no policies. It supports the same signal, method call and property. It has a membership certificate with PROVIDE|MODIFY on * bus objects. (i.e allow everything.) The device bus emits a signal. Signal API should give permission denied error. (no membership cert. found on device bus.) Device bus makes a method call, get property, set property. They should return error. (no membership cert. found on device bus.) When a peer bus emits a signal, the signal will be emitted successfully but will not be received by device bus. (Device bus will discard the signal as it does not have a membership certificate.) When a peer bus makes a method call, the method reply sent by the device bus will be an error message indicating permission denied. (Denied by policy by device bus.) When a peer calls GetProperty, SetProperty, the methods will get an error message indicating permission denied. (Denied by policy by device bus.)	**Yes**
Test the PROVIDE action mask for a method call, property, signal for ANY_USER	A device bus has PROVIDE action on a signal, method call and a property for ANY_USER. Another peer bus has no policies. It supports the same signal, method call and property. It has a membership certificate with PROVIDE|MODIFY on * bus objects. (i.e allow everything.) The device bus emits a signal. Signal API should give permission denied error. (no membership cert. found on device bus.) Device bus makes a method call, get property, set property. They should return error. (no membership cert. found on device bus.) When a peer bus emits a signal, the signal will be emitted successfully but will not be received by device bus. (Device bus will discard the signal as it does not have a membership certificate.) When a peer makes a method call, get property, set property they will be successful.	**Yes**

Test Objective	Procedure and Verification	Applicable for Standard client and Thin client
Test the OBSERVE action mask for a method call, property, signal for ANY_USER	A device bus has OBSERVE action on a signal, method call and a property for ANY_USER. Another peer bus has no policies. It supports the same signal, method call and property. It has a membership certificate with PROVIDE|MODIFY on * bus objects. (i.e allow everything.) The device bus emits a signal. Signal API should give permission denied error. (no membership cert. found on device bus.) Device bus makes a method call, get property, set property. They should return error. (no membership cert. found on device bus.) When a peer bus emits a signal, the signal will be emitted successfully but will not be received by device bus. (Device bus will discard the signal as it does not have a membership certificate.) When a peer makes a method call, error will be sent back as reply. (Denied by policy by device bus.) When a peer calls GetProperty, the property will fetched successfully. When a peer calls SetProperty, the methods will get an error message. (Denied by policy by device bus.)	**Yes**
Test the MODIFY action mask for a method call, property, signal for ANY_USER	A device bus has MODIFY action on a signal, method call and a property for ANY_USER. Another peer bus has no policies. It supports the same signal, method call and property. It has a membership certificate with PROVIDE|MODIFY on * bus objects. (i.e allow everything.) The device bus emits a signal. Signal API should give permission denied error. (no membership cert. found on device bus.) Device bus makes a method call, get property, set property. They should return error. (no membership cert. found on device bus.) When a peer bus emits a signal, the signal will be emitted successfully but will not be received by device bus. (Device bus will discard the signal as it does not have a membership certificate.) When a peer makes a method call, Get property, set property they will be successful.	**Yes**

Test Objective	Procedure and Verification	Applicable for Standard client and Thin client
Test the PROVIDE action mask for a method call, property, signal for SPECIFIC_USER. The peer bus is a SPECIFIC_USER.	A device bus has PROVIDE action on a signal, method call and a property for SPECIFIC_USER Another SPECIFIC_USER peer bus has no policies. It supports the same signal, method call and property. It has a membership certificate with PROVIDE|MODIFY on * bus objects. (i.e allow everything.) The device bus emits a signal. Signal API should give permission denied error. (no membership cert. found on device bus.) Device bus makes a method call, get property, set property. They should return error. (no membership cert. found on device bus.) When a peer bus emits a signal, the signal will be emitted successfully but will not be received by device bus. (Device bus will discard the signal as it does not have a membership certificate.) When a peer makes a method call, get property, set property, they will be successful.	**Yes**
Test the OBSERVE action mask for a method call, property, signal for SPECIFIC_USER. The peer bus is a SPECIFIC_USER.	A device bus has OBSERVE action on a signal, method call and a property for SPECIFIC_USER. Another SPECIFIC_USER peer bus has no policies. It supports the same signal, method call and property. It has a membership certificate with PROVIDE|MODIFY on * bus objects. (i.e allow everything.) The device bus emits a signal. Signal API should give permission denied error. (no membership cert. found on device bus.) Device bus makes a method call, get property, set property. They should return error. (no membership cert. found on device bus.) When a peer bus emits a signal, the signal will be emitted successfully but will not be received by device bus. (Device bus will discard the signal as it does not have a membership certificate.) When a peer makes a method call, error will be sent back as reply. (Denied by policy by device bus.) When a peer calls GetProperty, the property will fetched successfully. When a peer calls SetProperty, the method will get an error message. (Denied by policy by device bus.)	**Yes**

Test Objective	Procedure and Verification	Applicable for Standard client and Thin client
Test the MODIFY action mask for a method call, property, signal for SPECIFIC_USER. The peer bus is a SPECIFIC_USER	A device bus has MODIFY action on a signal, method call and a property for SPECIFIC_USER Another SPECIFIC_USER bus has no policies. It supports the same signal, method call and property. It has a membership certificate with PROVIDE|MODIFY on * bus objects. (i.e allow everything.) The device bus emits a signal. Signal API should give permission denied error. (no membership cert. found on device bus.) Device bus makes a method call, get property, set property. They should return error. (no membership cert. found on device bus.) When a peer bus emits a signal, the signal will be emitted successfully but will not be received by device bus. (Device bus will discard the signal as it does not have a membership certificate.) When a peer makes a method call, GetProperty, SetProperty, they will be successful.	**Yes**
Test the PROVIDE action mask for a method call, property, signal for SPECIFIC_USER. The peer bus is a NON_SPECIFIC_USER	A device bus has PROVIDE action on a signal, method call and a property for SPECIFIC_USER Another NON SPECIFIC_USER peer bus has no policies. It supports the same signal, method call and property. It has a membership certificate with PROVIDE|MODIFY on * bus objects. (i.e allow everything.) The device bus emits a signal. Signal API should give permission denied error. (no membership cert. found on device bus.) Device bus makes a method call, get property, set property. They should return error. (no membership cert. found on device bus.) When a peer bus emits a signal, the signal will be emitted successfully but will not be received by device bus. (Device bus will discard the signal as it does not have a membership certificate.) When a peer makes a method call, get property, set property error will be sent back as reply. (Denied by policy device bus as the user is not specific.)	**Yes**

Test Objective	Procedure and Verification	Applicable for Standard client and Thin client
Test the OBSERVE action mask for a method call, property, signal for SPECIFIC_USER. The peer bus is a NON_SPECIFIC_USER	A device bus has OBSERVE action on a signal, method call and a property for SPECIFIC_USER. Another NON SPECIFIC_USER peer bus has no policies. It supports the same signal, method call and property. It has a membership certificate with PROVIDE|MODIFY on * bus objects. (i.e allow everything.) The device bus emits a signal. Signal API should give permission denied error. (no membership cert. found on device bus.) Device bus makes a method call, get property, set property. They should return error. (no membership cert. found on device bus.) When a peer emits a signal, the signal be emitted successfully but will not be received by device bus. (Device bus will discard the signal as it does not have a membership certificate.) When a peer makes a method call, get property, set property, error will be sent back as reply. (Denied by policy by device bus.) When a peer calls GetProperty, error will be sent back as reply. (Denied by policy device bus as the user is not specific.) When a peer calls SetProperty, the methods will get an error message. (Denied by policy by device bus.)	**Yes**
Test the MODIFY action mask for a method call, property, signal for SPECIFIC_USER. The peer bus is a NON_SPECIFIC_USER	A device bus has MODIFY action on a signal, method call and a property for SPECIFIC_USER Another NON SPECIFIC_USER bus has no policies. It supports the same signal, method call and property. It has a membership certificate with PROVIDE|MODIFY on * bus objects. (i.e allow everything.) The device bus emits a signal. Signal API should give permission denied error. (no membership cert. found on device bus.) Device bus makes a method call, get property, set property. They should return error. (no membership cert. found on device bus.) When a peer emits a signal, the signal be emitted successfully but will not be received by device bus. (Device bus will discard the signal as it does not have a membership certificate.) When a peer makes a method call, GetProperty, SetProperty, error will be sent back as reply. (Denied by policy device bus as the user is not specific.)	**Yes**

Test Objective	Procedure and Verification	Applicable for Standard client and Thin client
Test the PROVIDE action mask for a method call, property, signal for GUILD based user.	A device bus has PROVIDE action on a signal, method call and a property for GUILD user. Membership authorization for GUILD: PROVIDE | MODIFY on all bus objects (use "*") Membership authorization applied on device bus. Another peer bus has no policies. It supports the same signal, method call and property. It has a membership certificate with PROVIDE|MODIFY on * bus objects. (i.e allow everything.) Device bus emits a signal. The signal will emitted successfully and received successfully. Device bus makes a method call, GetProperty, SetProperty. They should return error. (Denied by no policy on the peer bus.) When a peer emits a signal, the signal be emitted successfully and received successfully. When a peer makes a method call, get property, set property they will be successful.	**Yes**
Test the OBSERVE action mask for a method call, property, signal for GUILD based user.	A device bus has OBSERVE action on a signal, method call and a property for GUILD user Membership authorization for GUILD: PROVIDE | MODIFY on all bus objects (use "*") Membership authorization applied on device bus. Another peer bus has no policies. It supports the same signal, method call and property. It has a membership certificate with PROVIDE|MODIFY on * bus objects. (i.e allow everything.) Device bus emits a signal. The signal will emitted successfully and received successfully. Device bus makes a method call, GetProperty, SetProperty. They should return error. (Denied by no policy on the peer bus.) When a peer emits a signal, the signal be emitted successfully and received successfully. When a peer makes a method call, error will be sent back as reply. (Denied by policy device bus.) When a peer calls GetProperty, it will be successful. When a peer calls SetProperty, it will get an error message. (Denied by policy device bus.)	**Yes**

Test Objective	Procedure and Verification	Applicable for Standard client and Thin client
Test the MODIFY action mask for a method call, property, signal for GUILD based user.	A device bus has MODIFY action on a signal, method call and a property for GUILD user. Membership authorization for GUILD: PROVIDE | MODIFY on all bus objects (use "*") Membership authorization applied on device bus. Another peer bus has no policies. It supports the same signal, method call and property. It has a membership certificate with PROVIDE|MODIFY on * bus objects. (i.e allow everything.) Device bus emits a signal. The signal will be emitted successfully and received successfully. Device bus makes a method call, GetProperty, SetProperty. They should return error. (Denied by no policy on the peer bus.) When a peer emits a signal, the signal be emitted successfully and received successfully. When a peer makes a method call, get property, set property they will be successful.	Yes
For Signals, the policy does not take effect at all. Only the membership auth. data takes effect.	Emitter, Receiver : DENY policy for ANY_USER for a signal. Emiiter: Membership cert with PROVIDE, Receiver: Membership cert with OBSERVE. Verify that signal sent can be received by received successful.	Yes
Signal emitter has a DENY action in auth data.	A signal emitter application has no policies. The application has a membership certificate that has DENY action on the signal. Verify that Signal cannot be sent by the application.	**Yes**
Signal emitter has a OBSERVE action in auth data.	A signal emitter application has no policies. The application has a membership certificate that has OBSERVE action on the signal. Verify that Signal cannot be sent by the application.	**Yes**
Signal emitter has a MODIFY action in auth data.	A signal emitter application has no policies. The application has a membership certificate that has MODIFY action on the signal. Verify that Signal cannot be sent by the application.	**Yes**
Signal receiver has OBSERVE action on auth data.	Emitter, Receiver : No policies Emiiter: Membership cert with PROVIDE, Receiver: Membership cert with OBSERVE. Verify that signal sent can be received by received successful.	**Yes**

Test Objective	Procedure and Verification	Applicable for Standard client and Thin client
Signal receiver has PROVIDE action on auth data.	Emitter, Receiver : No policies Emiiter: Membership cert with PROVIDE, Receiver: Membership cert with PROVIDE Verify that signal can be sent but it wont be received by receiver as the receiver policy doesnt allow it to receive.	**Yes**
Signal receiver has MODIFY action on auth data.	Emitter, Receiver : No policies Emiiter: Membership cert with PROVIDE, Receiver: Membership cert with MODIFY Verify that signal sent can be received by received successful.	**Yes**
Signal receiver has DENY action on auth data.	Emitter, Receiver : No policies Emiiter: Membership cert with PROVIDE, Receiver: Membership cert with DENY Verify that signal can be sent but it wont be received by receiver as the receiver policy doesnt allow it to receive.	Yes
Sender having a MODIFY action on auth data for a method call, property.	Sender has no policies. It has MODIFY auth data for a method call, property. Receiver has MODIFY policy on a method call, property for a GUILD. Sender makes a method call, calls get/set property and gets replies from the receiver.	Yes
Sender having a PROVIDE action on auth data for a method call, property.	Sender has no policies. It has PROVIDE auth data for a method call, property. Receiver has MODIFY policy on a method call, property for a GUILD. Sender makes a method call and it gets error. (For PROVIDE auth data, sender cannot make method calls.) Sender calls get/set property and gets replies from the receiver.	Yes
Sender having a OBSERVE action on auth data for a method call, property	Sender has no policies. It has OBSERVE auth data for a method call. Receiver has MODIFY policy on a method call, property for a GUILD. Sender makes a method call and gets an error from the sender side. The local auth data (which has OBSERVE) prevents the method call from going out. Sender calls get property and it is successful. Sender calls set property and gets an error from the sender side. The local auth data (which has OBSERVE) prevents the set property call from going out.	Yes
Sender having a DENY action on auth data for a method call, property.	Sender has no policies. It has DENY auth data for a method call, property. Receiver has MODIFY policy on a method call, property for a GUILD. Sender makes a method call, calls get/set property and gets an error from the sender side. The local auth data (which has DENYprevents the method call, propety calls from going out.	Yes

Test Objective	Procedure and Verification	Applicable for Standard client and Thin client
Receiver of a method call, property does not check its local auth data when it receives a method call, property requests.	Sender has no policies. It has MODIFY auth data for a method call, property for "guild1". Receiver has MODIFY policy on a method call, property for a GUILD "guild1". Receiver has a DENY auth data for a method call, property for "guild1". Sender makes a method call, get property/set property and is successful. (The receiver does not check its local auth data. Thus the DENY rule on its auth data does not take effect.)	Yes
Policy guild id at receiver and auth. data guild id of the sender do not match.	Sender has no policies. It has MODIFY auth data for a method call, property for "guild1". Receiver has MODIFY policy on a method call, property for a GUILD "guild2". Receiver has a DENY auth data for a method call, property for "guild2". Sender makes a method call, get property/set property and it gets error. (The receiver policy guild id (guild2) does not match with sender auth data (guild1).	Yes
Multiple GUILDs scenario.	GUILD1: PROVIDE|MODIFY on signal 1. GUILD2: PROVIDE|MODIFY on signal 2. Device bus has no polices, It belongs to GUILD1, GUILD2. Consumer bus 1: No policies, belongs to GUILD1. Consumer bus 2: No polices, belongs to GUILD2. Device emits both signals. Verify that signal 1 is received only by consumer 1. SIgnal 2 is received only by consumer 2.	Yes
Empty policy means implicit deny.	Device bus: Implements a method call. Has an empty policy. Peer bus has no policies installed. Peer bus makes a method call. The method call is denied at the device bus's end.	**Yes**
No policy means implicit deny.	Device bus: Implements a method call. Has no policies installed. Device bus is claimed. Peer bus has no policies installed. Peer bus makes a method call. The method call is denied at the device bus's end.	**Yes**
RemoveMembership should remove membership authorization data.	A device bus has MODIFY|PROVIDE action on a signal, method call and a property for GUILD user. Membership authorization for GUILD: PROVIDE | MODIFY on all bus objects (use "*") Membership authorization applied on device bus. Another peer bus has no policies. It supports the same signal, method call and property. It has a membership certificate with PROVIDE|MODIFY on * bus objects. (i.e allow everything.) The device bus emits a signal. The signal will be emitted successfully.	**Yes**

Test Objective	Procedure and Verification	Applicable for Standard client and Thin client
	Device bus makes a method call, GetProperty, SetProperty. They will be successful. Admin bus calls RemoveMembership on device bus. The device bus emits a signal. The signal will not be sent. (Denied because the device bus does not have any membership certificates.) Device bus makes a method call. Method call will be denied. (Denied because the device bus does not have any membership certificates.) Device bus calls GetProperty, SetProperty. They will be denied. (Denied because the device bus does not have any membership certificates.)	
Reset the device after Installing policies and Membership certificates.	Claim a device bus, Install policies. Install membership certficates, auth data. Check for PermissionMgmt.Notification. Check that policy serialNumber and claimed state. Reset the device bus. Check for PermissionMgmt.Notification. Check that policy serialNumber=0 and claimed state and "unclaimed"	**Yes**
Keystore size should not grow upon multiple claim/install polices/install membership certifctaes/reset.	Claim a device bus, Install policies. Install membership certficates, auth data. Check Keystore size. Reset the device bus. Check Keystore size. Repeat above two operations 1000 times. KeyStore size should not have a growth because of claim/reset operations.	**Yes**
Memory profiling test	Run unit tests under valgrind. No memory leaks should be found.	**Yes**
Delegating a membership certifcate with delegate=true	admin claims mid-user bus, consumer bus. mid-user bus claims provider bus. Consumer bus has no policies. It supports signal1 and signal2. It has a membership certificate with PROVIDE|MODIFY on * bus objects. (i.e allow everything.) Admin installs membership auth data with MODIFY|PROVIDE policy on signal1 and signal2 on the mid-user bus. Admin installs membership certificate (delegate=true) and membership auth data with MODIFY|PROVIDE policy on signal1 and signal2 on the consumer bus. mid-user bus installs membership certificate (delegate=false) and membership auth data with MODIFY|PROVIDE policy on signal1 on the provider bus. (This should include the membership auth data from the admin bus also, forming a chain.) Provider bus emits signal1. It is received successfully by the consumer. Provider bus emits signal2. It wont send the signal.	**Yes**

Test Objective	Procedure and Verification	Applicable for Standard client and Thin client
Delegating a membership certifcate with delegate=false	admin claims mid-user bus, consumer bus. mid-user bus claims provider bus. Consumer bus has no policies. It supports signal1 and signal2. It has a membership certificate with PROVIDE|MODIFY on * bus objects. (i.e allow everything.) Admin installs membership auth data with MODIFY|PROVIDE policy on signal1 and signal2 on the mid-user bus. Admin installs membership certificate (delegate=false) and membership auth data with MODIFY|PROVIDE policy on signal1 and signal2 on the consumer bus. mid-user bus installs membership certificate (delegate=false) and membership auth data with MODIFY|PROVIDE policy on signal1 on the provider bus. (This should include the membership auth data from the admin bus also, forming a chain.). (This method call should fail because the membership cert. installed by admin had delegate=false.) Provider bus emits signal1. It wont send the signal. Provider bus emits signal2. It wont send the signal.	**Yes**
Peers using RSA mode of authentication should not be able to send messages IF policies are installed.	Two peers are claimed and are using security 2.0 with policies installed. Peers set up a RSA session between themselves. Peers cannot send or receive messages as the trust establishment is not enforced with the RSA mode of authentication.	**No**
Peers using RSA mode of authentication should be able to send messages after Reset.	Two peers are claimed and are using security 2.0 with policies installed. Peers set up a RSA session between themselves. Peers cannot send or receive messages as the trust establishment is not enforced with the RSA mode of authentication. Admin resets the buses. Peers set up a RSA session between themselves. Peers CAN send or receive encrypted messages.	**No**
Admin user has no restrictions.	A device bus implements a method call and registers for a signal. Admin bus claims the device bus and has an empty policy installed on it. Admin bus makes a method call. Method call should be successful. Admin bus emits a signal. The signal should be received by the device bus.	**Yes**

Test Objective	Procedure and Verification	Applicable for Standard client and Thin client
If a member or interface is not secure, there are no policy enforcements.	A device bus implements an unsecure interface that supports a method call, signal and property. A peer bus also implements the same unsecure interface. Both the buses have DENY action on a policy on the interface. They also have membership certificates that has a DENY action on the interface. device bus makes a method call, get property, set property on the peer bus. They should be successful and unencrypted. device bus sends a signal. The signal will be received by the peer bus. The signal will be unencrypted. Peer bus makes a method call, get property, set property on the device bus. They should be successful and unencrypted. Peer bus sends a signal. The signal will be received by the device bus. The signal will be unencrypted.	**Yes**
Admin installs Membership cert. signed by a different bus (bus1).	On provider: Admin bus installs Membership cert signed by bus1. On consumer: Admin bus installs Membership cert signed by bus1. Provider and consumer should be able to talk. This is because they both are issued certificate by bus 1. Bus1 is NOT the admin.	
Admin installs Membership cert. signed by a different buses(bus1 and bus2).	On provider: Admin bus installs Membership cert signed by bus1. On consumer: Admin bus installs Membership cert signed by bus2. Provider and consumer should NOT be able to talk. This is because they both are issued certificate by different buses. Bus1 and bus2 are not admin.	

3.4 End-to-End test, Endianness and reboot-based test

Test Objective	Procedure and Verification	Applicable for Standard client and Thin client
End-to-end test. This involves one OpenWRT (Big Endian) and 1 Linux (Little Endian) machine. The test also verifies that after reboot, the state of the system does not change as all the policies are stored in the keystore file.	OpenWRT runs an application1 that emits a signal. application1 has a membership cert. to emit the signal under GUILD type. Linux machine runs thin client application2 that emits a signal. application2 has a membership cert. to emit the signal under GUILD type. Linux machine runs admin app. Admin app claims application1, application2. Admin installs policies on application1, application2. Admin installs membership certificates on application1, application2. Linux machine runs application 3 that receives the signals sent by application1 and application2. application1 and application2 emits signals. Verify that application3 successfully receives both the signals. Reboot the Linux machine and the OpenWRT router. After reboot, Launch application1, application2 and application3. Do not launch admin application. application1 and application2 emits signals. Verify that application3 successfully receives both the signals.	**No**

3.5 Application Manifest

Test Objective	Procedure and Verification
Verify the PermissionConfigurator APIs	Load the PermissionConfigurator object using GetPermissionConfigurator() Call SetClaimableState(false); Call GetClaimableState and verify the state is false. Check for PermissionMgmt.Notification.claimable=Unclaimable SetClaimableState(true) Call GetClaimableState and verify the state is true. Check for PermissionMgmt.Notification.claimable=Claimable Call GetSigningPublicKey. Verify that the public key is not set. Call GenerateSigningKeyPair. check for ER_OK. Call GetSigningPublicKey. Verify that the public key is set now and is not empty.

Test Objective	Procedure and Verification
SetManifest and GetManifest	Create a rule: e.g.: MODIFY action for a method call. SetPermissionManifest on the following rule. Call org.allseen.Security.PermissionMgmt.Notification.GetManifest on the above bus. Verify that the manifest fetched has the same rules that was set.
Reset the device bus using PermissionConfigurator	Claim a device bus, Install policies. Install membership certficates, auth data. check for PermissionMgmt.Notification. Check that policy serialNumber and claimed state. Reset the device bus using PermissionConfigurator. Check for PermissionMgmt.Notification. Check that policy serialNumber=0 and claimed state and "unclaimed"
Get the peer public key for an SRP based connection.	Establish a SRP session between two peers. Call GetConnectedPeerPublicKey on the other peer. This should return error. Error should indicate that the connection is not ECDHE_DSA based.
Get the peer public key for an ECDHE_NULL based connection.	Establish a ECDHE_NULL session between two peers. Call GetConnectedPeerPublicKey on the other peer. This should return error. Error should indicate that the connection is not ECDHE_DSA based.
Get the peer public key for an ECDHE_DSA based connection.	Establish a ECDHE_DSA session between two peers. Call GetConnectedPeerPublicKey on the other peer. Verify that the public key of the other peer was fetched successfully..

3.6 Existing security mechanisms test for unclaimed apps

Test Objective	Procedure and Verification	Applicable for Standard client and Thin client
Test SRP mode of authentication between two peers. Will work only if NONE of the devices are claimed.	Two peers are NOT claimed. Peers set up an SRP session between themselves. Peers should be able to send/receive encrypted messages.	No
Test RSA mode of authentication between two peers. Will work only if NONE of the devices are claimed.	Two peers are NOT claimed. Peers set up an RSA session between themselves. Peers should be able to send/receive encrypted messages.	No
Test LOGON mode of authentication between two peers. Will work only if NONE of the devices are claimed.	Two peers are NOT claimed. Peers set up an LOGON session between themselves. Peers should be able to send/receive encrypted messages.	No

Test Objective	Procedure and Verification	Applicable for Standard client and Thin client
Test PINX mode of authentication between two peers. Will work only if NONE of the devices are claimed.	Two peers are NOT claimed. Peers set up an PINX session between themselves. Peers should be able to send/receive encrypted messages.	No
Test SRP mode of authentication between two peers after Claiming them.	Two peers are claimed but policies installed that will allow all mesaages. Peers set up an SRP session between themselves. Peers should NOT be able to send/receive encrypted messages.	No

3.7 Corrupt Keystore (ASC and ATC)

Procedure:

1. Admin bus claims device bus, installs policies, installs membership certificates and auth data.
2. Device bus is able to send and receive encrypted messages successfully from a peer bus.
3. Establish ECDHE_DSA session between admin bus and device bus.
4. Establish ECDHE_DSA session between non-admin bus and device bus.
5. Add random bytes to the keystore file of the device bus.
6. Admin calls "Claim" on device bus. It should return error indicating "invalid or corrupt keystore file"
7. Admin calls "InstallPolicy" on device bus. It should return error indicating "invalid or corrupt keystore file"
8. Admin calls "RemovePolicy" on device bus. It should return error indicating "invalid or corrupt keystore file"
9. Admin calls "Reset" on device bus. It should return error indicating "invalid or corrupt keystore file"
10. A non-admin bus calls "Claim" on device bus. It should return error indicating "invalid or corrupt keystore file"
11. A non-admin bus calls "InstallPolicy" on device bus. It should return error indicating "invalid or corrupt keystore file"
12. A non-admin bus calls "RemovePolicy" on device bus. It should return error indicating "invalid or corrupt keystore file"
13. A non-admin bus calls "Reset" on device bus. It should return error indicating "invalid or corrupt keystore file"
14. Verify that the device bus cannot send or receive messages to the peer bus. The error should indicate an "invalid or corrupt keystore file"

NOTE: The only way to recover is to delete the keystore externally and start all over again.

3.8 Certificate expiry

Test Objective	Procedure and Verification	Applicable for Standard client and Thin client
Test Identity certificate expiry	Admin installs Identity certificate on device bus with a validity period of 5 minutes Admin calls installs policy on the device bus. Admin waits for 6 minutes. Admin calls RemovePolicy on the device bus. This should be successful as the peer keys have not expired and hence trust establishment is not trigerred. Admin clears the peer keys Admin calls install policy on the device bus. This should trigger an ECDHE_DSA session which should fail as the device bus identity certificate is not valid.	**Yes**
Test Membership certificate expiry	Admin installs membership certificate on device bus with a validity period of 5 minutes device bus establishes a secure connection with a peer bus using ECDHE_DSA. It should succeed. device bus sends a signal to peer bus using a GUILD. The signal will be successfully received by peer bus. Device bus tears down the connection. device bus waits for 6 minutes. device bus establishes a secure connection with a peer bus using ECDHE_DSA. It should succeed. (Succeed because the identity certificate is still valid.) device bus sends a signal to peer bus using a GUILD. The signal will be rejected by peer bus because the identity certificate of the bus has expired.	**Yes**