FNV2 Security Architecture - In-Vehicle SOA security (FNV2-SecM-SOA) (0.1) (Copy)

Status	COMPLETE
Completion (%)	100%
Version	0.1
Owner	Unknown User (elin12)
Reviewer	Depooter, Stephen (S.)
Review Date	02 May 2017
Approver	Hukku, Rajiv (R.)
Approve Date	14 May 2017

1. Glossary

Term	Definition
ACL	Access Control List
ADAS	Advanced Driver Assistance Systems
CA	Certificate Authority
ECU	Electronic Control Unit
HUD	Heads Up Display
IPC	Inter-Process Communication
IPMA	Image Processing Module ADAS
MQTT	Message Queue Telemetry Transport
SOA	Service Oriented Architecture
TLS	Transport Layer Security

2. Summary

The FNV2 In-vehicle SOA middleware provides a framework for different local and remote components in the in-vehicle network to exchange data in a secure and efficient manner. The SOA middleware is based on the MQTT publish-subscribe messaging protocol which runs on top of TCP/IP. TLS should be used to secure the TCP/IP network traffic over the in-vehicle Ethernet. Each client of the SOA middleware should be securely identifiable. Access of each MQTT topic may be restricted by an access control mechanism to ensure only authorized clients have access to lists of topics that they have permissions to access. The MQTT broke should also monitor for traffic abnormalities and log security instances for analysis.

3. Requirements

Requirement ID and JIRA	Description	HLD Section	How we meet	Implementation JIRAs
REQ_SOA_SEC.01 ECG-2599	MQTT Authentication process must be protected from an attacker listening on the wire	4.1	Use TLS sessions to protect the MQTT communications on the Ethernet TLS RSA key pair should be provisioned by Key Management	To be implemented by SOA team: SOA HLD Security Team: Key storage ECG-1915
REQ_SOA_SEC.02 ECG-2600	Access to MQTT Topics must have Access Control applied per user	4.3	client identifier is used in MQTT topic for access control	To be implemented by SOA team: SOA HLD
REQ_SOA_SEC.03 ECG-2601	A service must be able to authenticate which client sent a given request. This cannot use client provided data in the payload.	4.2	Use uid/gid to identify client applications	To be implemented by SOA team: SOA HLD Security Team: UID/GID management ECG-71
REQ_SOA_SEC.04 ECG-2602	RPC operations over the SOA system must not allow a third-party to spoof a reply to a request.	4.2	Use uid/gid in RPC operations to identify sender of messages	To be implemented by SOA team: SOA HLD
REQ_SOA_SEC.05 ECG-2603	The location of a client on a Sync/TCU/ECG should not matter from an authentication point of view.	4.2	The local gateway on Sync /TCU will attach local client's identifier in MQTT messages. The location of the client has no impact on access control.	To be implemented by SOA team: SOA HLD
REQ_SOA_SEC.06 ECG-2658	SOA traffic should be monitored for abnormal behavior, security instances should be logged	4.4	MQTT broker should monitor traffic and log security instances.	ECG-2757

4. High Level Design

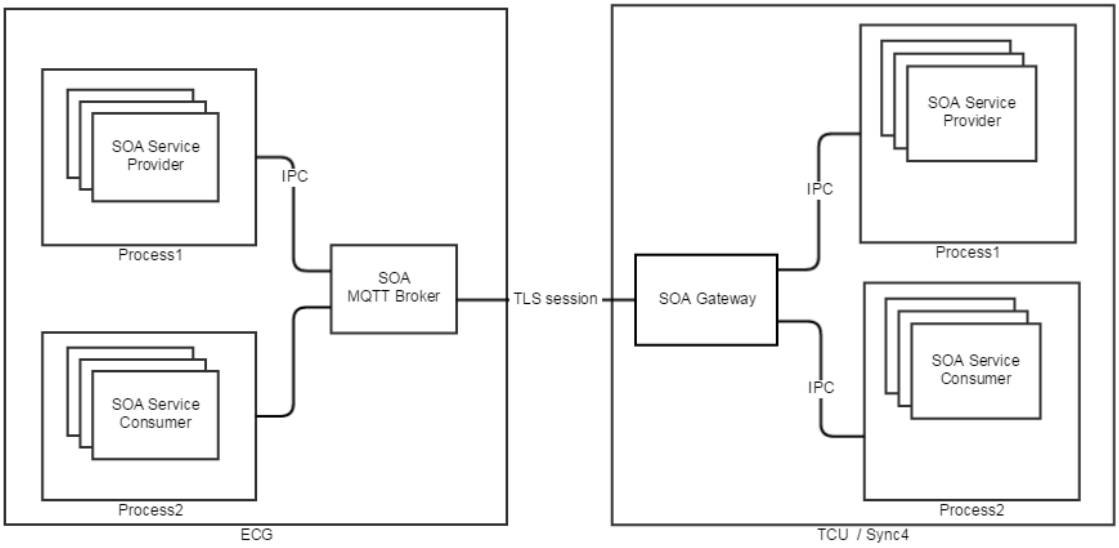
4.1. Secure SOA traffic with TLS Session

4.1.1. TLS session between SOA MQTT broker and SOA gateway

The SOA system is built on top of MQTT. The MQTT broker may have multiple listeners. When the broker receives traffic over Ethernet, it must authenticate that the sender and the receiver are authentic FNV2 network nodes. This means that we require a 2-way TLS handshake which will provide mutual authentication of the network devices using RSA signatures. This is not necessary for traffic to the MQTT broker that occurs over localhost traffic on the ECG. If the traffic never hits an Ethernet PHY, we don't require a TLS handshake or encryption. Since each module has a RSA private key, this key may not be accessible to arbitrary code to use when establishing a TLS session to the broker. For SOA clients external to ECG (i.e. processes running in TCU or SYNC4), the TCU or SYNC4 unit shall implement a SOA gateway which functions as a proxy between the SOA service provider/consumer processes. The SOA gateway should have access to the RSA private key and can establish a TLS session with the ECG MQTT broker and carry out two-way authentication.

The CN (commonName) element from the TLS certificate can be used as the MQTT username. No password needs to be provisioned for each username because it is sufficient for a client to present a valid TLS certificate for authentication.

The overall SOA architecture for secured connection between ECG and TCU/SYNC4 is depicted in the diagram below. More detailed design of the SOA gateway is documented in FNV2-SOA Security Architecture (Copy). This HLD focuses on the detailed security requirement for the "TLS session" between the "SOA MQTT Broker" and the "SOA Gateway".



The current document mainly addresses the SOA communications between TCU, Sync and ECG. However, the same secure communication design should applied on future ECUs with Ethernet connections to the ECG SOA broker, such as IPMA, Instrument Cluster, HUD, etc.

4.1.2. TLS connection crypto configurations

4.1.2.1. TLS Version

The MQTT broker and the SOA gateway shall be configured to use **TLS v.1.2**.

4.1.2.2. TLS cipher suite selections

TLS standard supports a large selection of key exchange, encryption and message authentication algorithms. For our SOA implemention, one or multiple of the following cipher suites should be supported.

- TLS_ECDHE_RSA_WITH_AES_256_CBC_SHA384_P384
- TLS_ECDHE_RSA_WITH_AES_128_CBC_SHA256_P256
- TLS_ECDHE_RSA_WITH_AES_256_CBC_SHA_P384
- TLS_ECDHE_RSA_WITH_AES_128_CBC_SHA_P256

The sequence of names in the cipher suite represents: key exchange algorithm, bulk encryption algorithm, message authentication code (MAC) algorithm, and the elliptic curve. For example, TLS_ECDHE_RSA_WITH_AES_256_CBC_SHA384_P384 represents using ECDHE-RSA as key exchange, AES_256_CBC for bulk encryption, SHA384 for MAC, and the P-384 elliptic curve.

The selected cipher suites provide good strength of security and forward secrecy.

4.1.2.3. TLS digital certificate

TLS digital certificate should use the X.509 format digital certificate with RSA-2048 key and the sha256withRSAEncrytion Signature Algorithm. Each policy area (ECG, SYNC, TCU) will have a policy CA which issues the intermediate CA certificate for that policy area. Partial certificate chain validation will carried out up to the policy CA's intermediate certificate.

4.1.2.4. Key storage and management

ECG: The SOA MQTT Broker on ECG will need the following crypto keys/certificates:

- RSA-2048 key pair:

  ECG_SOA_Auth_RSA-2048_cert: certificate containing ECG public key, signed by Ford ECG intermediate CA with ECG_SOA_CA_RSA-cert
  ECG_SOA_Auth_RSA-2048_priv_key: ECG private key corresponding to above public key

- SYNC_SOA_CA_RSA-cert: Ford SYNC CA certificate
- TCU_SOA_CA_RSA-cert: Ford TCU CA certificate

SYNC: The SOA Gateway on Sync will need the following crypto keys/certificates:

- RSA-2048 key pair:

  SYNC_SOA_Auth_RSA-2048_cert: certificate containing Sync public key, signed by Ford Sync intermediate CA with SYNC_SOA_CA_RSA-cert
  SYNC_SOA_Auth_RSA-2048_priv_key: Sync private key corresponding to above public key

- ECG_SOA_CA_RSA-cert: Ford ECG CA certificate

TCU: The SOA gateway on TCU will have the following crypto keys/certificates:

- RSA-2048 key pair:

  TCU_SOA_Auth_RSA-2048_cert: certificate containing TCU public key, signed by Ford TCU intermediate CA with TCU_SOA_CA_RSA-cert
  TCU_SOA_Auth_RSA-2048_priv_key: TCU private key corresponding to above public key

- ECG_SOA_CA_RSA-cert: Ford ECG CA certificate

More details about the key storage and management are contained in Section 4.1.7, Section 4.3.2 and Section 4.7.1 of ECG Key Storage and Management.

4.2. Client identification using uid/gid

Local clients on SYNC/TCU send SOA messages through the local SOA gateway. The local SOA gateway should have a secure mechanism to uniquely identify each client application. For each application, it will be running with an assigned unique uid/gid after dropping root. There are several secure ways for the SOA gateway to retrieve a client's credentials, depending on the chosen IPC (Inter Process Communication) between SOA gateway and clients.

- For IPC using Unix Domain Socket(**AF_UNIX**) on Linux, or Local Domain Socket(**AF_LOCAL**) on QNX, the **getsockopt()** API with the **SO_PEERCRED** (Linux) or **LOCAL_CREDS** (QNX) option can provide the SOA gateway the client credentials, which include the client's **uid/gid**.
- For QNX systems, if the SOA gateway runs as a resource manager, it can use the **iofunc_client_info_able()** or **iofunc_client_info_ext()** APIs to retrieve the **_client_info** structure. _client_info contains the **_cred_info** structure which provides the real and effective uid/gid information of the client.

Currently, the preference is use the Unix Domain Socket as the IPC to minimize the implementation differences between Linux and QNX.

Client credentials from these API are provided by the QNX/Linux kernel, not filled by the client itself. So, they can be reliably used by the SOA gateway to identify the uid/gid of the connecting client.

Management of uid/gid are described in FNV2-SecM-PLTFRM UID/GID Assignment (Copy). Sync and ECG may also allow installation of new applications. We will need to assign new uid/gids to each new app. For system security, the /etc/passwd and /etc/group files which control uid/gid allocations will be locked down on the read-only file system, and does not allow adding new uid/gid at runtime. But the /etc/passwd and /etc/group files can be pre-configured to have a pool of pre-defined unused uid/gids, which can then be assigned to newly installed apps. The pre-defined uid /gids may also be pre-configured with different security access levels.

Using uid/gid as client identification credentials requires the client applications to drop root and run with the lower privileged user credentials. But many legacy applications on Sync are currently running as root. Dropping root should be achievable for most applications by properly retaining necessary QNX abilities or Linux capabilities. For applications which still needs to run as root, they can be put into one group of trusted applications with uid of 0, which will have high access permissions.

Application's uid/gid can be combined with a nodeID which identifies ECG, SYNC or TCU to get a unique client identifier across the different components, which is described in more detail at FNV2-SOA Security Architecture (Copy)

4.3. SOA topic access control

The location of the access permission check can be done at the ECG broker, the local SOA gateway or the service provider. The preferred method is to authorize the access permission in a central location at the ECG broker. Each SOA MQTT topic should be tagged with a client identifier as defined in 4.2, which can then be used to verify whether a given client is allowed access to a service from each service provider. The proposed topic structure is defined in a format as "**.../<ServiceType>/<clientIdentifier>/...**", which is documented in more detailed at FNV2-SOA Security Architecture (Copy).

4.4. SOA traffic monitoring and logging

SOA traffic should be monitored for abnormal behavior, and security related events should be logged. The event logs can be sent to the Ford Analytics for further analysis.

- Traffic monitoring: the SOA MQTT broker should implement necessary counters to record basic statistics of SOA traffic. These statistics can help to debug problems and also identify potentially compromised MQTT clients. Examples of traffic patterns to monitor may include:

    1. number of successful/failed connection attempts by a client
    2. number of publish/subscribe messages sent from each MQTT client
    3. traffic throughput from each client

- Logging: the SOA MQTT broker should maintain a log of important events for security auditing. Example of events as candidates for logging may include:

    1. connection requests from a client
    2. failed authentications with invalid TLS credentials
    3. client timeout or disconnections
    4. RPC requests which lead to denied service access based on ACL rules
    5. RPC transactions involving security sensitive information

5. Performance

TLS crypto operations require additional computational overhead. Chipset solutions from different venders provide various hardware crypto engines or dedicated hardware blocks for offloading crypto operations. Evaluations should be done to find the most suitable method to reduce memory and CPU overhead on each target platform.

6. Open Questions/Issues