ETSI GR MEC 041 V3.1.1 (2024-03)

Multi-access Edge Computing (MEC); Study on MEC Security

Disclaimer

The present document has been produced and approved by the Multi-access Edge Computing (MEC) ETSI Industry Specification Group (ISG) and represents the views of those members who participated in this ISG.
It does not necessarily represent the views of the entire ETSI membership.

Reference
DGR/MEC-0041v311MECSecurity

Keywords
application, MEC, security

ETSI

650 Route des Lucioles
F-06921 Sophia Antipolis Cedex - FRANCE

Tel.: +33 4 92 94 42 00 Fax: +33 4 93 65 47 16

Siret N° 348 623 562 00017 - APE 7112B
Association à but non lucratif enregistrée à la
Sous-Préfecture de Grasse (06) N° w061004871

Contents

Intellectual Property Rights

Essential patents

IPRs essential or potentially essential to normative deliverables may have been declared to ETSI. The declarations pertaining to these essential IPRs, if any, are publicly available for **ETSI members and non-members**, and can be found in ETSI SR 000 314: *"Intellectual Property Rights (IPRs); Essential, or potentially Essential, IPRs notified to ETSI in respect of ETSI standards"*, which is available from the ETSI Secretariat. Latest updates are available on the ETSI Web server (https://ipr.etsi.org/).

Pursuant to the ETSI Directives including the ETSI IPR Policy, no investigation regarding the essentiality of IPRs, including IPR searches, has been carried out by ETSI. No guarantee can be given as to the existence of other IPRs not referenced in ETSI SR 000 314 (or the updates on the ETSI Web server) which are, or may be, or may become, essential to the present document.

Trademarks

The present document may include trademarks and/or tradenames which are asserted and/or registered by their owners. ETSI claims no ownership of these except for any which are indicated as being the property of ETSI, and conveys no right to use or reproduce any trademark and/or tradename. Mention of those trademarks in the present document does not constitute an endorsement by ETSI of products, services or organizations associated with those trademarks.

DECT™, **PLUGTESTS™**, **UMTS™** and the ETSI logo are trademarks of ETSI registered for the benefit of its Members. **3GPP™** and **LTE™** are trademarks of ETSI registered for the benefit of its Members and of the 3GPP Organizational Partners. **oneM2M™** logo is a trademark of ETSI registered for the benefit of its Members and of the oneM2M Partners. **GSM**® and the GSM logo are trademarks registered and owned by the GSM Association.

Foreword

This Group Report (GR) has been produced by ETSI Industry Specification Group (ISG) Multi-access Edge Computing (MEC).

Modal verbs terminology

In the present document "**should**", "**should not**", "**may**", "**need not**", "**will**", "**will not**", "**can**" and "**cannot**" are to be interpreted as described in clause 3.2 of the ETSI Drafting Rules (Verbal forms for the expression of provisions).

"**must**" and "**must not**" are **NOT** allowed in ETSI deliverables except when used in direct citation.

1 Scope

The present document outlines security topics and paradigms that apply to MEC deployments across the realms of application/platform security and zero-trust architecture. The present document considers prior work of other standards bodies and industry associations. It identifies gaps in ETSI ISG MEC specifications and provides recommendations for new normative work.

2 References

2.1 Normative references

Normative references are not applicable in the present document.

2.2 Informative references

References are either specific (identified by date of publication and/or edition number or version number) or non-specific. For specific references, only the cited version applies. For non-specific references, the latest version of the referenced document (including any amendments) applies.

NOTE: While any hyperlinks included in this clause were valid at the time of publication, ETSI cannot guarantee their long term validity.

The following referenced documents are not necessary for the application of the present document but they assist the user with regard to a particular subject area.

[i.1] ETSI GR MEC 001: "Multi-access Edge Computing (MEC); Terminology".

[i.2] ETSI GR MEC 035: "Multi-access Edge Computing (MEC); Study on Inter-MEC systems and MEC-Cloud systems coordination".

[i.3] CNCF: "Cloud Native Computing Foundation".

[i.4] NIST SP 800-207: "Zero Trust Architecture", August 2020.

[i.5] SDP Specification 1.0: "Software Defined Perimeter", Cloud Security Alliance (2015).

[i.6] ETSI White Paper No. 46: "MEC security: Status of standards support and future evolutions", 2nd edition - September 2022.

[i.7] TCG: "Trusted Computing Group".

[i.8] IETF RFC 9334: "Remote Attestation Procedures".

[i.9] GSMA White Paper: "Operator Platform Telco Edge Proposal", Oct. 2020.

[i.10] GSMA OPG.02: "Operator Platform: Requirements and Architecture"; Version 5.0, 26th July 2023.

[i.11] ETSI GS MEC 003: "Multi-access Edge Computing (MEC); Framework and Reference Architecture".

[i.12] ETSI GS MEC 009: "Multi-access Edge Computing (MEC); General principles, patterns and common aspects of MEC Service APIs".

[i.13] Trusted Platform Module Library: "Part 1: Architecture".

[i.14] IETF RFC 6749: "The OAuth 2.0 Authorization Framework".

[i.15] IETF RFC 8705: "OAuth 2.0 Mutual-TLS Client Authentication and Certificate-Bound Access Tokens".

[i.16] IETF RFC 7591: "OAuth 2.0 Dynamic Client Registration Protocol".

[i.17] IETF RFC 2986: "PKCS #10: Certification Request Syntax Specification Version 1.7".

[i.18] IETF CoRIM: "Concise Reference Integrity Manifest".

[i.19] NTIA SBOM: "SBOM at a glance".

[i.20] 3GPP TR 33.848: "Study on Security Impacts of Virtualisation", Version 18.0.0.

[i.21] ETSI GS NFV-SEC 012: "System architecture specification for execution of sensitive NFV components", Version 3.1.1.

[i.22] IETF draft-fossati-tls-attestation: "Using Attestation in Transport Layer Security (TLS) and Datagram Transport Layer Security (DTLS)".

[i.23] TCG: "DICE Endorsement Architecture for Devices", Version 1.0, Revision 0.38.

[i.24] IETF draft-sandowicz-httpbis-httpa2: "The Hypertext Transfer Protocol Attestable (HTTPA) Version 2".

[i.25] Linux IMA: "Integrity Measurement Architecture".

[i.26] NIST SP 800-218: "Secure Software Development Framework (SSDF) Version 1.1", February 2022.

[i.27] ISO/IEC 5962:2021: "Software Package Data Exchange".

[i.28] SLSA: "Supply chain Levels for Software Artifacts".

[i.29] OWASP CycloneDX: "Lightweight Bill of Materials (BOM) standard".

[i.30] ETSI GS MEC 010-2: "Multi-access Edge Computing (MEC); MEC Management; Part 2: Application lifecycle, rules and requirements management".

[i.31] ETSI GS NFV-SEC 013: "Network Functions Virtualisation (NFV) Release 3; Security; Security Management and Monitoring specification".

[i.32] 3GPP TR 33.894: "Study on zero-trust security principles in mobile networks", Version 18.0.0.

[i.33] Cloud Security Alliance: "Cloud Controls Matrix", Version 3.0.1.

[i.34] GSMA: "FS.31 GSMA Baseline Security Controls", Version 2.0.

[i.35] ETSI GS NFV-SEC 013: "Network Functions Virtualisation (NFV) Release 3; Security ; Security Management and Monitoring specification".

[i.36] ETSI GS NFV-SEC 024: "Network Functions Virtualisation (NFV) Security; Security Management Specification".

[i.37] TCG DICE: "Device Identifier Composition Engine", Level 00 Revision 78.

[i.38] TCG DICE-Layering: "DICE Layering Architecture", Version 1.0, Revision 0.19.

[i.39] W3CDID: "Decentralized Identifiers", Version 1.0.

[i.40] ETSI GS MEC 011: "Multi-access Edge Computing (MEC); Edge Platform Application Enablement".

[i.41] ETSI GS MEC 002: "Multi-access Edge Computing (MEC); Use Cases and Requirements".

3 Definition of terms, symbols and abbreviations

3.1 Terms

For the purposes of the present document, the terms given in ETSI GR MEC 001 [i.1] apply.

3.2 Symbols

Void.

3.3 Abbreviations

For the purposes of the present document, the abbreviations given in ETSI GR MEC 001 [i.1] and the following apply:

AA	Authentication and Authorization
CNCF	Cloud Native Computing Foundation
CSP	Cloud Service Provider
DoS	Denial of Service
DDoS	Distributed Denial of Service
RoT	Root of Trust
SMM	Security Monitoring and Management
TEE	Trusted Execution Environment
ZTA	Zero-Trust Architecture

4 Overview

4.1 Introduction

The present document outlines security topics and paradigms that apply to MEC deployments across the realms of App/platform security and zero-trust architecture.

The present clause introduces the MEC security landscape, with reference to current initiatives in the field. Clause 5 documents key issues that illustrate security challenges in MEC systems to analyse gaps in the current ETSI MEC specifications. Clause 6 summarizes the recommendations from clause 5 for closing those gaps. The present document considers prior work of other industry/standards bodies as well as all relevant work done in ETSI.

4.2 Platform and Software Security

Edge computing platforms are often located outside physical security zones typical for data centres. This makes the critical importance of platform integrity [i.6] an increasingly top-of-mind issue for edge cloud architects. ETSI ISG MEC needs to be able to address questions around whether platform integrity needs to play a role in processes such as App LCM and if so, what that role is and what, if any, API enhancements are needed to enable it.

Further, edge deployments constitute a complex multi-vendor, multi-supplier, and multi-stakeholder ecosystem lacking a central entity that implements system-wide security assurances or accepts full liability if things go wrong. Software assets from these various parties along with their user's data are under threat of IP theft, ransomware attacks or Denial of Service (DoS) attacks. ETSI MEC specifications could thus draw upon threat mitigation strategies and practices currently employed by major Cloud Service Providers (CSPs) and/or driven by the Cloud Native Computing Foundation (CNCF) [i.3].

4.3 Zero-Trust Architecture

The migration towards cloud-native implementations of both Apps and network functions combined with the highly distributed and multi-tenant nature of distributed clouds is increasingly driving adoption of Zero-Trust Architecture (ZTA) [i.4] concepts such as Software Defined Perimeter [i.5]. Because ZTA is premised on validating trust on each and every cloud entity (e.g. App instance), it is not unreasonable to expect that adoption of ZTA may impact MEC App LCM operations, which may require additional API standardization.

4.4 Trusted Computing and Attestation

Trusted computing technologies uphold consistent behaviour of computing systems (e.g. MEC Hosts) by ensuring that their underlying components are unmodified and not executing unauthorized or malicious code. This is achieved during the boot process by certifying the provenance of firmware and successive levels of the software stack through a chain of trust originating at a root of trust. The Trusted Computing Group (TCG) [i.7] specifications outline the requirements and cryptographic constructions for implementing trusted computing as a vendor-neutral standard.

Attestation is the process of creating, conveying, and appraising the trustworthiness characteristics of a computing system. This is accomplished through *Attester* and *Verifier* roles in a scenario where a *Relying Party* (e.g. API endpoint) assesses the trustworthiness of another computing entity (e.g. API requester). The Verifier may reside on (for example) the API backend and the Attester on (for example) the API requester. The Relying Party, typically, issues a challenge request to the Attester for a specific scope of the API requester's characteristics. Integrity measurements (e.g. digests) of system software components (e.g. firmware, kernel modules) from the API requester's environment may be compiled into *Attestation Evidence* by an Attester and presented to a Verifier. The Verifier authenticates the received Attestation Evidence and compares it to known good values previously delivered to the Verifier to arrive at a verdict on the trustworthiness of the attesting system and its software stack. Typically, the Attester is constructed with a root of trust that is implicitly trusted, that is to say, trust in the root of trust is vouched for by its manufacturer by issuing a certificate or other endorsement document that describes the root of trust technology. Typically, a root of trust is implemented using tamper-resistant technology (see [i.37]). A trustworthy Attester typically has bootstrapped trusted modules that are inspected by the root of trust or a delegate trusted module, see [i.38]. Further, trust relationships between the entities that implement attestation roles, e.g. Attester, Verifier and Relying Party, are typically established using Public Key Infrastructure (PKI) but alternatives are also possible, see [i.39].

IETF RFC 9334 [i.8] defines Attestation roles, conceptual messages, and patterns for message exchange that can be integrated into network protocols and APIs. The aim of attestation integration is to condition traditional data and control flow operations based on a trust assessment. Traditional message flow assumes the peers of a message exchange are implicitly trustworthy without performing a trust assessment.

The decentralized nature of cloud-based, virtualized architectures has made it easier for firmware and kernel rootkits to cause privilege escalation and exfiltrate data. Popular CSPs may rely on trusted computing technology in their infrastructures to detect and eliminate such attacks. As MEC deployments are subject to similar attack vectors, it may be useful for MEC designers to consider adoption of trusted computing methodologies for future MEC standards.

4.5 Security Monitoring and Management

Security Monitoring and Management (SMM) refers to activities undertaken by network operators or communication service providers employing a set of tools and mechanisms to continually preserve the security of their network deployment (physical, virtualized or hybrid).

In particular, security monitoring involves using passive or active probing of traffic flows on the user data, control, or management planes, and extracting statistics of infrastructure usage (network, compute, storage) of the network components. At a higher level, security monitoring can involve inspection of network entity state and message flows between network entities. Importantly, security monitoring has to lead to a view over the entire network (virtual and physical).

Security management on the other hand involves two aspects: the establishment of security policies, and the enforcement of these policies. Notably, there is a processing of the data gathered from the security monitoring tools, often aided by AI/ML and data analytics. In addition, the enforcement of policies involves mainly automated (as opposed to human-decision-driven) security management actions. Numerous tools exist commercially for both security monitoring and management (separately or bundled together).

Security monitoring and management systems can help to ensure platform and software security (clause 4.2) of the MEC platform and other MEC components. A SMM system employs behavioural monitoring (also referred to as telemetry) to identify MEC entities acting outside expectations for proper and secure operation. Once identified, the SMM system can take steps to limit the operation (and hence potential incurred damage) of the identified MEC entity.

In support of SMM, MEC data collection and processing is a key activity, e.g. logging of all security events experienced by a MEC App, MEC host, MEC Platform, and other MEC components. Collection and processing of such data can be done in an efficient but secure fashion with privacy considerations in mind, and the logs can be access-controlled to prevent unauthorized modification and ensure accuracy of subsequent auditing or forensics activities. Anomalies found in collected security-related data could indicate a malfunction or a security incident (e.g. malware has been activated).

For security management, a security manager entity will take action in response to a suspect incident, behaviour or activity.

The SMM for a MEC system SMM can be defined so as to integrate with existing tools being used in the operator networks, such as Network Intrusion Detection Systems (NIDS), Network Intrusion Prevention Systems (NIPS) and deep packet inspection firewalls. Operators manage complex networks and use a variety of tools to undertake their SMM activities. The industry and standards bodies have not produced a unified SMM authoritative framework or general usage document, but best practices and guidance exist, allowing the operators to select and customize the set of tools to fit their own business objectives. Therefore, a MEC system, once added to an existing network, can provide the requisite security monitoring telemetry feeds and allow for security management actions to be taken in response, based on policies defined by each operator.

Other industry forums and SDOs have laid out some best practices and high-level measures for security monitoring and/or management (e.g. it is part of Zero Trust currently under study in 3GPP TR 33.894 [i.32], part of the "Cloud Control Matrix v3.0.1" [i.33] in the Cloud Security Alliance, and part of FS.31 Baseline Security Controls v2.0 [i.34] in GSMA). In particular, SMM for NFV systems is specified in ETSI GS NFV-SEC 013 [i.31], with the goal to establish the NFV Security Lifecycle Management and Security Monitoring requirements and architecture. Some elements of the ETSI NFV-SEC defined SMM can be re-used for ETSI MEC (where applicable). These can be taken into account by a MEC system operator when defining the particular profiles and policies for the various MEC entities and integrate them into their existing network management tools.

4.6 MEC Federations

ETSI MEC is introducing architectural enhancements to enable a federation of MEC systems [i.2], allowing for inter-MEC system communications in scenarios involving multiple MNOs. This is a use case where security is a critical requirement from operators and all federating entities. Each MNO establishes and administers its own security domain that should remain isolated from the security domains of other MNOs.

The GSMA Operator Platform Group (GSMA OPG) has defined the notion of an Operator Platform (OP) [i.9], [i.10] as the entity that mediates access to the edge cloud capabilities of an MNO. The GSMA further stipulates that OP instances implement the East/West Bound Interface (E/WBI) between MNOs where the OP abstracts away the internal topology and configuration of the MNO. For this purpose, GSMA OPG introduces a Common Data Model (CDM) to describe characteristics of the elements of an OP instance, including OP roles, applications, edge clouds, and security. The GSMA OPG requirements as inferred from [i.9] suggest the ability to identify clients, application providers and applications, along with encrypted and authorized access to edge resources across federated OP instances.

As ETSI MEC is considering GSMA OPG requirements for realizing MEC Federations, the security aspects of OP architecture and related MEC implications are to be carefully considered. In particular, MEC security can be studied from the perspective of edge cloud services through an analysis of its threat models and providing solutions for managing identity, authentication and authorization of client and edge applications across edge clouds via their OP instances.

4.7 MEC architecture and security

4.7.1 Description

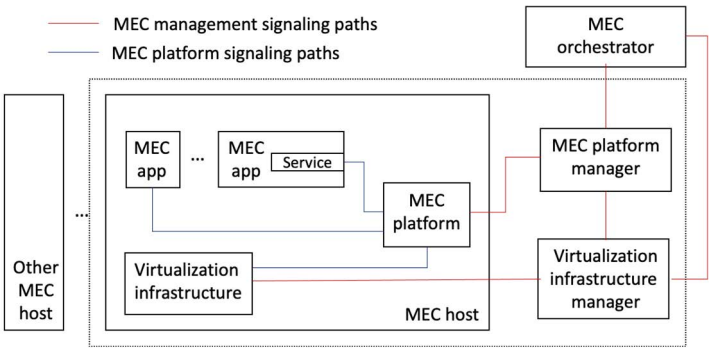


Figure 4.7.1-1: Multi-access edge system

Figure 4.7.1-1 is a simplified depiction of the MEC reference architecture that is documented in ETSI GS MEC 003 [i.11]. A MEC system broadly consists of MEC hosts and management elements to support MEC applications. MEC applications are software-only entities that execute in Virtual Machines (VM) or containers over a virtualization infrastructure that is situated close to end-users of the applications.

A MEC host is a physical entity that contains a MEC platform and the virtualization infrastructure. MEC management operates through the MEC platform to configure the deployment of MEC applications to the virtualization infrastructure and to aid them throughout their lifecycle.

MEC management at the system-level includes the MEC Orchestrator (MEO) which maintains a view of the whole MEC system. The MEO onboards application packages, performs integrity checks, and selects an appropriate MEC host to deploy the application.

MEC management at the host-level includes the MEC Platform Manager (MEPM) and the Virtualization Infrastructure Manager (VIM). The MEPM serves as the control plane for the MEC platform and is a conduit to the MEO. The VIM is responsible for allocating, managing, and releasing virtualized (computing, storage, and networking) resources of the virtualization infrastructure.

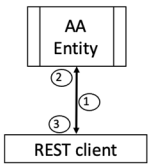
ETSI GS MEC 009 [i.12] describes patterns for authentication and authorization of MEC applications over RESTful and alternate transports that are summarized below in clauses 4.7.2 and 4.7.3. These patterns also apply to the NFV variant of the MEC reference architecture specified in ETSI GS MEC 003 [i.11].

4.7.2 Authentication

MEC specifications mandate HTTPS support in all RESTful MEC service APIs using TLS v1.2 or TLS v1.3 and prohibit the use of HTTP without TLS or TLS versions preceding v1.2. TLS secures the transmission channel between peers. During the TLS handshake, peers may mutually authenticate themselves using X.509 certificates that follow a chain of trust to a trusted Certificate Authority.

4.7.3 Authorization

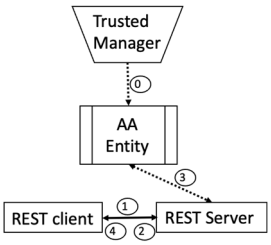
Authorization to access RESTful MEC service APIs defined by ETSI ISG MEC is achieved using the OAuth 2.0 protocol [i.14]. The MEC specifications posit an Authentication and Authorization (AA) entity that is accessible to both the REST client and server. A trusted manager entity configures the AA entity with appropriate credentials and REST clients' access rights to server resources. REST clients further authenticate to the AA entity to request an access token using the OAuth client credentials grant.



1. REST client initiates a HTTPS session with the AA entity
2. REST client issues a request to its OAuth token endpoint providing its client secret
3. The AA entity returns an access token to the Client

Figure 4.7.3-1: Acquiring an access token using OAuth 2.0

REST clients present this token in requests to resources hosted in REST servers over an authenticated HTTPS session. On receiving a request, REST servers may communicate with the AA entity to authorize the request before returning a response.



0. A Trusted Manager authorizes requests from the REST client
1. REST client initiates a HTTPS session with the REST server
2. REST client issues a request including its access token as the bearer token
3. REST server contacts the AA entity to authorize the request
4. If the request is authorized, the REST server returns a response

Figure 4.7.3-2: Authorization of requests to MEC service using an OAuth 2.0 access token

The trusted manager entity, its methods for configuring REST clients' access rights, the AA entity and the interactions between the REST server and the AA entity are all outside the scope of MEC specifications.

If OAuth is not supported over an alternate transport, requests from the REST client can be authorized using its client credential that was presented during the TLS handshake.

5 Key issues and potential solutions

5.1 Key issue #1: Stolen MEC App access tokens

5.1.1 Description

MEC applications are designed using varying engineering practices employed by individual vendors. Sensitive data like credentials or access tokens may be persisted to a data store for subsequent recall and such implementation details might differ between MEC applications. The evolving nature of software invariably exposes vulnerabilities which are routinely exploited by attackers to steal sensitive data even in the presence of other security measures.

Backups and replication of data store segments make it more difficult to track leakages of sensitive data. While data may be stored in encrypted form, compromised encryption keys by a malicious insider or a privilege escalation attack in the edge cloud would break the assumed protections. MEC systems are conceived as multi-tenant and multi-stakeholder environments and thus present a large attack surface.

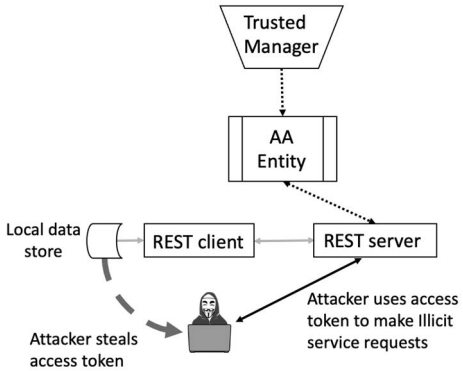


Figure 5.1.1-1: Illicit service requests using stolen access token

Figure 5.1.1-1 illustrates the risk of REST servers conferring access to protected resources to an adversary who presents a stolen OAuth access token as the bearer token over requests. Such a replay of access tokens by unauthorized parties could be prevented by binding access tokens to the applications' TLS credentials.

5.1.2 Solution proposal #1: Adopt OAuth 2.0 Mutual-TLS Client Authentication and Certificate-Bound Access Tokens

5.1.2.1 Description

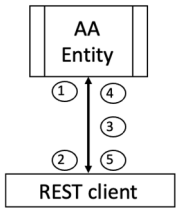
Access to MEC service APIs is secured using HTTPS, but mutual TLS authentication of MEC applications remains optional. Mutual TLS authentication enforces verification of a client's identity over an established chain of trust. In deference to this goal, IETF RFC 8705 [i.15] specifies mutual TLS authentication using X.509 certificates as a possible authentication method for registered OAuth clients to access the token endpoint in authorization servers. It further lays out a methodology for authorization servers to bind access tokens with clients' X.509 certificates such that OAuth protected resources and/or their associated authorization servers can verify the requesting clients' identities as a mandatory step in processing service requests. Therefore, clients that do not belong to the trust chain or do not assert the identity bound to a presented access token would be denied access to protected resources and thereby mitigate this threat.

5.1.2.2 Configure client credentials

MEC applications should acquire X.509 certificates signed by a trusted Certificate Authority (CA) to use as the credential while negotiating TLS sessions. A MEC application may generate a public/private key pair and store its private key in a secure location. Over a secure protocol with the CA, the MEC application requests and receives a X.509 identity certificate for its public key. This procedure is based on messages and practices specified in Public Key Cryptography Standards (PKCS) (IETF RFC 2986 [i.17]). MEC applications subsequently use this certificate to authenticate themselves to both the authorization server (i.e. the AA entity) and REST servers.

5.1.2.3 OAuth Client registration and access token

The REST client is a confidential client as defined by IETF RFC 6749 [i.14] and is issued a X.509 client certificate as described in clause 5.1.2.2. This facilitates mutual-TLS authentication of its OAuth role with the AA entity which (as opposed to the OAuth shared-secret authentication method) allows the AA entity to issue certificate-bound access tokens as specified by IETF RFC 8705 [i.15]. Consequently, authorizations to OAuth protected resources in REST servers would become contingent on the REST client using its X.509 identity certificate to establish HTTPS sessions with REST servers.



1. REST client issues a registration request to the AA entity with its metadata
2. AA entity returns a client identifier (client_id) following a successful registration
3. REST client negotiates a mutually authenticated HTTPS session with the AA entity using its identity certificate
4. REST client issues a request to the AA entity's OAuth token endpoint providing its client_id
5. AA entity matches the client identity from its certificate and its configuration and returns an OAuth access token

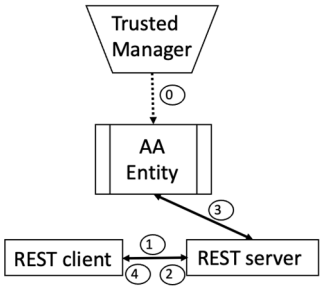
Figure 5.1.2.3-1: OAuth Client registration and authorization request

Before initiating the OAuth protocol, the REST client should register itself with the AA entity to receive an OAuth client identifier (client_id). IETF RFC 7591 [i.16] defines sequences for registering OAuth 2.0 clients with authorization servers. This is captured in steps 1 and 2 in figure 5.1.2.3-1. The REST client issues a request to the AA entity's OAuth registration endpoint containing a series of client metadata values (client credentials grant type, application name, description, etc.).

As the REST client will use OAuth 2.0 mutual-TLS client authentication, its registration request should specify *tls_client_auth* as its OAuth token endpoint authentication method (IETF RFC 8705 [i.15], Section 2.1.1). It should additionally include the *tls_client_auth_subject_dn* parameter with a value that matches the subject distinguished name of the REST client's identity certificate (IETF RFC 8705 [i.15], Section 2.1.2). Following a successful registration, the AA entity returns a client_id to the REST client.

Through steps 3-5 in figure 5.1.2.3-1, the REST client requests and receives an OAuth access token in accordance with IETF RFC 8705 [i.15]. In a mutually authenticated HTTPS session with the AA entity, the REST client issues a request to the OAuth token endpoint supplying its client_id. The AA entity locates the client configuration by client_id and verifies that the REST client's X.509 certificate used during the TLS handshake matches the expected certificate (IETF RFC 8705 [i.15], Section 2). It then proceeds to issue the REST client an OAuth access token bound to this certificate as specified in IETF RFC 8705 [i.15], Section 3. The REST client may persist the access token for further use.

5.1.2.4 Authorizing access to MEC Services



0. A Trusted Manager authorizes requests from the REST client
1. REST client negotiates a mutually authenticated HTTPS session with the REST server
2. REST client issues a request to the REST server providing its OAuth access token as the bearer token
3. REST server via the AA entity verifies the client<->token binding and the client's authorization
4. If the request is authorized, the REST server returns a response

Figure 5.1.2.4-1: MEC Service authorization using certificate bound access tokens

Authorizations of client requests are performed as specified in IETF RFC 8705 [i.15] and depicted in figure 5.1.2.4-1. As conceived in ETSI GS MEC 009 [i.12], a trusted manager configures appropriate permissions granting the REST client access to protected resources. The REST client establishes a mutually authenticated HTTPS session with the REST server. It makes a service request by supplying its access token as the bearer token. On receiving the request, the REST server communicates with the AA entity to verify the binding between the REST client's X.509 certificate and the access token (IETF RFC 8705 [i.15], Section 3.2) and to validate the REST client's access rights for this request. If the request is authorized, the REST server returns a response to the REST client.

5.1.3 Evaluation

As the TLS handshake provides a proof-of-possession of the clients' private keys that are assumed to be securely held, it would follow that a mutually authenticated client carrying an identity that is bound to the presented access token is most likely the authorized party. Thus, employing OAuth 2.0 mutual-TLS client authentication and certificate-bound access tokens enhances the security of OAuth protected resources because a stolen access token would alone be insufficient to authorize requests in the absence of the client credential.

ETSI ISG MEC may consider the following recommendation:

- Prescribe pattern in ETSI GS MEC 009 [i.12] for OAuth authorization using mutual-TLS client authentication and certificate-bound access tokens (IETF RFC 8705 [i.15]).

5.2 Key issue #2: Stolen MEC App identity

5.2.1 Description

As described in clause 5.1.1, varying design practices and software vulnerabilities expose a large attack surface making it hard to identify and track the leakage of sensitive data such as authentication credentials. A limited interface to an application's identity and cryptographic keys is critical without which an adversary may circumvent other protections bestowed by authentication and authorization protocols.

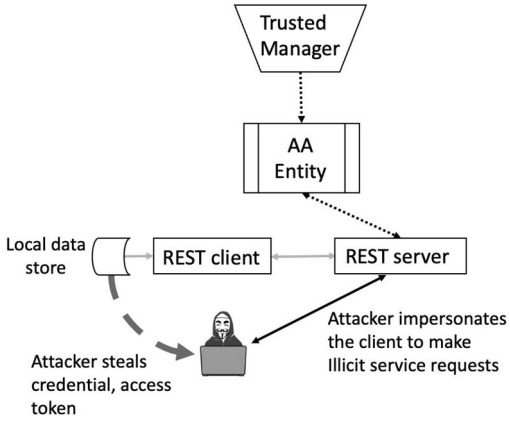


Figure 5.2.1-1: Threat of impersonation, identity theft

Figure 5.2.1-1 illustrates the threat of a stolen identity by an attacker who can successfully authenticate to the REST server masquerading as the REST client and access protected resources. This threat could be mitigated by tightly binding a single secure instance of the application's identity and credential(s) to the application.

5.2.2 Solution proposal #1: Enhance MEC App Authentication using hardware-based security

5.2.2.1 Description

Dedicated cryptographic hardware in MEC hosts can be leveraged to serve as a Root of Trust (RoT) for applications. Secure crypto processors perform various cryptographic functions including key generation, encryption, decryption, signing and verifying digital signatures. They are hardened with security measures which make them tamper resistant and protect cryptographic keys such that they never leave the RoT entity. As the keys are inaccessible to software, they are far less susceptible to compromise.

Hardware-based crypto services are accessed via a limited and well-defined API over a trusted path. Thus, they expose a smaller attack surface. RoT entities can be validated for their authenticity and security capabilities and are designed to provide verifiable assertions, as known as attestation, that a cryptographic key is resident in, and protected by, the entity. Attestation increases confidence in the protection of the information secured by the key.

IETF RFC 9334 [i.8] defines a conceptual model, roles, and patterns to implement remote attestation. This is the mechanism by which one party (the *Attester*) produces cryptographically signed claims (the *Evidence*) about its trustworthiness to enable a peer (the *Relying Party*) to decide whether to consider the attester as trustworthy. An additional party (the *Verifier*) appraises evidence from the attester and generates *Attestation Results* to assist relying parties in their decision process. The verifier utilizes *Endorsements* from an *Endorser* (e.g. the manufacturer of the RoT and other trusted components) when evaluating the evidence. These roles may be realized in MEC systems, anchored by a RoT entity, to better secure MEC applications.

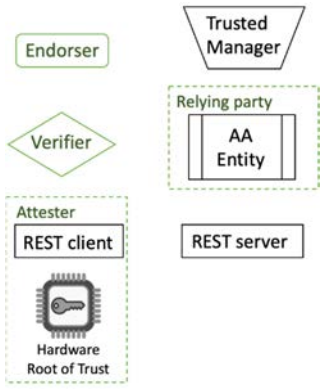
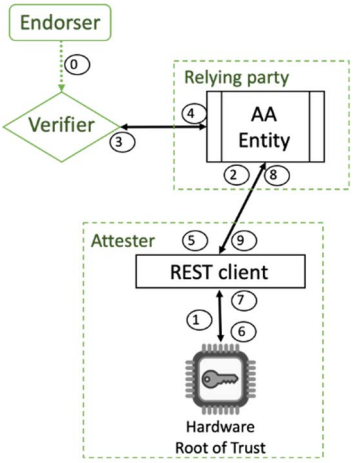


Figure 5.2.2.1-1: Hardware root-of-rust and attestation roles

Figure 5.2.2.1-1 shows MEC system entities with attestation roles overlays (in green) where the attestation roles (attester, relying party, verifier, and endorser) are specified by IETF RFC 9334 [i.8]. In this scenario, the REST client is bound to a RoT entity that takes on the attester role. In practice, this binding may be achieved via a Virtual Machine Manager (VMM) or a Trusted Execution Environment (TEE) in the MEC host. The attester produces evidence asserting that security relevant assets, such a cryptographic keys and trusted software are protected by an appropriate environment. The verifier appraises attester evidence, and based on a positive attestation result, the AA entity (as known as relying party) issues a credential (X.509 certificate) to the REST client that certifies its key as described in clause 5.2.2.2. The REST client uses this credential to proceed with OAuth2 token registration and issuance.

5.2.2.2 Generating Client Credentials bound to the RoT entity

The REST client should first establish a trust relationship with the AA entity. It is assumed that the REST client is provisioned with the AA entity's certificate trust anchor (as known as public key) for the purposes of establishing a secure (HTTPS) session with the AA entity. The AA entity further establishes trust in the REST client by obtaining a remote attestation of the REST client. The AA entity then issues an identity credential to the trusted REST client that is protected by its RoT. This is achieved through a series of steps illustrated in figure 5.2.2.2-1 that are based on principles and methods outlined in the Trusted Platform Module Library specification [i.13]. A similar conceptual flow applies to other RoT technologies and standards.



0. Endorser(s) (e.g. manufacturer) provides endorsements to verify authenticity of attesters
1. Attester generates an Attestation key
2. Attester presents evidence of the attestation key to the AA entity
3. AA entity forwards evidence to the Verifier
4. Verifier evaluates evidence based on endorsements and reference values and generates attestation results
5. AA entity certifies the attestation key based on attestation results
6. Attester generates an unrestricted signing key
7. Attester produces proof of the key signed by the attestation key as evidence
8. Attester presents said evidence to the AA entity to request for a X.509 identity certificate
9. Following successful evaluation of the evidence, the AA entity returns a X.509 identity certificate

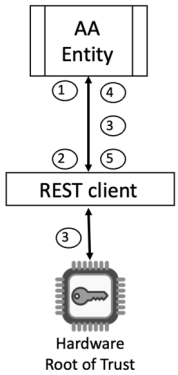
Figure 5.2.2.2-1: Generating a protected X.509 identity certificate

The REST client initiates this process by requesting an *Attestation Key* from its RoT. A special, pre-installed credential in the RoT entity (e.g. an *Endorsement Certificate*) vouches that this key is resident in the RoT, which also collects evidence about the TEE. The TEE measurements are included in the evidence that is presented to the AA entity. The AA entity invokes the verifier to appraise this evidence. The verifier utilizes endorsements and reference values to appraise TEE evidence. Reference values establish a baseline for known good state of the TEE that is compared to measurements of the TEE that were included in the evidence. Evidence would have to remain unchanged from its known good state for the verifier to conclude that the TEE is trusted. The verifier then confirms that the RoT and related trusted entities are genuine and generates attestation results suitable for the AA entity (i.e. the relying party) to make trust-based decisions. If the AA entity certifies the attestation results, it acknowledges this in its response to the REST client.

The REST client then generates an unrestricted signing key to request the AA entity to issue a client credential for this key. For this it collects proof of the key signed by the attestation key as evidence to present to the AA entity. The attestation key is a restricted signing key that can only sign data that was generated within the RoT entity. As the AA entity had previously certified the attestation key, its signed evidence of the newly generated key proves that the key is resident in the RoT entity. Following a review of this evidence, the AA entity issues a X.509 identity certificate for the new key to the REST client. The REST client subsequently uses this credential to mutually authenticate to the AA entity and REST Servers.

5.2.2.3 OAuth Client registration and access token

Having acquired a credential secured by its R3oT entity (as described in clause 5.2.2.2), the REST client registers itself with the AA entity and further authenticates to the AA entity using its X.509 identity certificate to request for a certificate-bound access token following the procedures described in clause 5.1.2.3 and illustrated in figure 5.2.2.3-1.

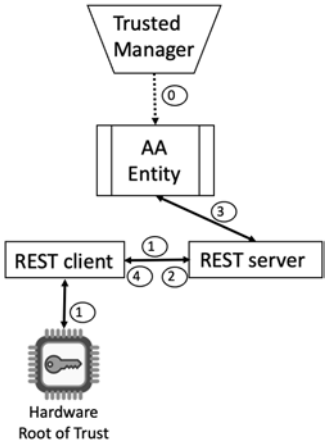


1. REST client issues a registration request to the AA entity with its metadata
2. AA entity returns a client identifier (client_id) following a successful registration
3. REST client negotiates a mutually authenticated HTTPS session with the AA entity using its identity certificate
4. REST client issues a request to the AA entity's OAuth token endpoint providing its client_id
5. AA entity matches the client identity from its certificate and its configuration and returns an OAuth access token

Figure 5.2.2.3-1: OAuth Client registration and authorization request

5.2.2.4 Authorizing access to MEC Services

Authorization of client requests are carried out by the steps described in clause 5.1.2.4 and illustrated in figure 5.2.2.4-1.



0. A Trusted Manager authorizes requests from the REST client
1. REST client negotiates a mutually authenticated HTTPS session with the REST server
2. REST client issues a request to the REST server providing its OAuth access token as the bearer token
3. REST server via the AA entity verifies the client<->token binding and the client's authorization
4. If the request is authorized, the REST server returns a response

Figure 5.2.2.4-1: MEC Service authorization using certificate bound access tokens

5.2.3 Evaluation

The proposed solution (in clause 5.2.2) mitigates the threat of impersonation of MEC application instances by adversaries using stolen credentials. This is achieved by storing the credential in a hardened environment that is authenticated by a hardware root of trust. Attestation procedures aid in generating the credential through stepwise verifications that build the AA entity's trust in the MEC application instance and in the protection of its key. This improves the overall defensive posture of MEC applications as keys cannot be hijacked in application software.

5.3 Key issue #3: Compromised MEC applications, asset theft

5.3.1 Description

MEC (service producing and service consuming) applications are packaged into a virtual machine or as a containerized application. The MEC Orchestrator (MEO) on-boards applications and prompts virtualization infrastructure manger(s) on selected MEC host(s) to allocate virtualized resources to execute the application over the virtualization infrastructure.

Virtualized workloads might be compromised in a multitude of ways. Spear phishing attacks by adversaries, accidental misconfigurations in cloud services, or password/access token disclosure through code repositories are just a few instances of directly or indirectly leaking access to virtualized applications. Having obtained such access, an attacker may install rootkits or persistent malware into a virtual machine for exfiltrating sensitive data or copy a virtual machine snapshot to run a parallel instance. ETSI GS MEC 003 [i.11] specifies that the MEO checks for the integrity and authenticity of MEC application packages when they are onboarded. However, it does not elucidate further details and does not address threats that may arise during the MEC application lifecycle.

The edge computing paradigm coupled with artificial intelligence is poised to disrupt major industries including transportation and retail. Such use cases will elicit access to expansive user datasets by workloads executing on the edge. Any unmitigated data breaches would thus have disastrous outcomes and further put such technology out of compliance with international standards or regulations for security and privacy.

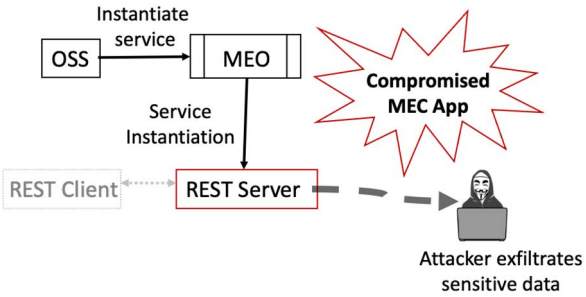


Figure 5.3.1-1: Attack through a compromised MEC service

Figure 5.3.1-1 depicts an example of user data/asset theft by an attacker through an undetected compromise of a virtual machine that is executing a service producing MEC application (i.e. the REST Server in the above figure). Identical threats apply to service consuming MEC applications.

To identify such concealed attacks, it is important to define and characterize trust in the context of MEC applications. Related standards and documents by 3GPP and ETSI NFV present their conceptions of trustworthiness for workloads in their respective domains. 3GPP TR 33.848 [i.20] discusses a threat like the above illustration in clause 5.14.2 and suggests that trust should be assessed for the top-level functional (VNF) layer and all layers below. Accordingly, it outlines the following security requirements in clause 5.14.3 to enable a 3GPP NF to make trust-based decisions about interacting with another VNF.

- It should be possible to attest a virtualized 3GPP NF through the full attestation chain from the hardware layer through the virtualization layer to the VNF layer.
- Attestation of a platform's integrity should be linked to the application layer and possible for other functions to query. If platform attestation fails, the virtualized 3GPP NF should not be allowed to run.
- Attestation of the VNF should be performed prior to deployment/network integration and during operations.
- Attestation of the VNF should be done at the hardware, virtualization and NF layers.

Separately, ETSI GS NFV-SEC 012 [i.21] surveys a wide range of host infrastructure technologies through which a *Trusted Computing Base (TCB)* may be established on which a workload can operate within defined levels of trust in the overall security of the system. These include the ability to provide evidence on demand of security claims relating to a host's security capabilities and integrity of the software layers executing on it both at provisioning time and at runtime. It further recommends that the host system supports the use of a service (external to the host) providing remote attestation.

In addressing the threat in figure 5.3.1-1, the solution in clause 5.3.2 of the present document also captures the above requirements and aims to draw upon ongoing collaborative efforts in the IETF [i.8], [i.18], [i.22], [i.24] and TCG [i.23] which would make it feasible to specify new APIs and design practices in the ETSI MEC specifications.

5.3.2 Solution proposal #1: Verify provenance of MEC applications through cryptographic attestations

5.3.2.1 Description

The unpredictable nature of attacks in distributed, multi-tenant software environments calls for a "never trust, always verify" (as known as *zero-trust*) approach to security where every user, data flow, device and application is deemed untrusted until (re-)certified for some measure of trustworthiness.

The National Institute of Standards and Technology (NIST) [i.26] has embraced Software Bill Of Materials (SBOM) [i.19] to better understand and manage risk across the software supply chain. The industry community is collaborating [i.27], [i.28], [i.29] to standardize data models for describing SBOMs and mapping SBOMs to known vulnerabilities to characterize threats faced by individual software components. An edge software vendor is expected to compile the software stack underlying a MEC application after performing due diligence and settling on a baseline configuration with known and acceptable risk. All trustworthiness claims regarding the MEC application would thus be relative to this baseline state at deployment time. Furthermore, after a MEC application is deployed and operational, *Trusted Computing* methods aided by a *Root of Trust* (RoT) can track deviations in the MEC application and its execution environment from their trusted baseline.

The PC/smartphone industry offers a precedent for this general model with *Verified Boot* (illustrated in figure 5.3.2.1-1) wherein layers of the software stack from firmware to the bootloader and kernel are signed and their hashes securely held in each layer. During the boot procedure, each layer computes a cryptographic hash of the next layer. The hash of the software of the next layer is assessed within each layer and if unchanged, that layer continues to load the next layer, being assured of its trustworthiness. The core RoT might exist in hardware or in the BIOS firmware. Any change in the hashes from their known good values would be considered untrustworthy and interrupt the boot cycle. This ensures that a device only ever boots software that is trusted by the Original Equipment Manufacturer (OEM). A similar verification regime can be adapted with further enhancements to networked and virtualized applications in edge clouds through remote attestation.

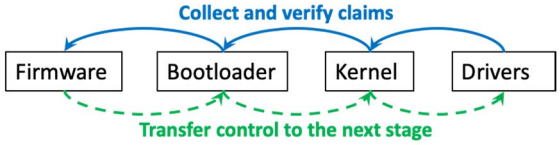


Figure 5.3.2.1-1: Illustration of the secure boot process

IETF RFC 9334 [i.8] outlines a conceptual framework for securely collecting trustworthiness claims about software components. A known entity (the *Attester*) authenticates those claims to present as *Evidence* to a *Verifier* which appraises the evidence against *Reference Values* (i.e. a trusted baseline state of those software components) and delivers *Attestation Results* to a *Relying Party* to make trust-based decisions. Thus, an attestation framework may also serve as a *Policy Enforcement Point (PEP)* in a MEC system for it to take suitable actions to mitigate an attack in the event of a compromise. In the context of MEC applications, a possible response to a negative assessment of its trust might be for the MEO to suspend the application.

In IETF RFC 9334 [i.8], the attester role (depicted in figure 5.3.2.1-2) is defined as containing at least one *Attesting Environment* and *Target Environment*. The attesting environment is an immutable or otherwise tamper-resistant component that collects claims about the target environment. The target environment is measured (for e.g. through cryptographic hashing) by the attesting environment. Examples of attesting environments include Trusted Execution Environments (TEEs), Trusted Platform Modules (TPMs) or BIOS firmware.

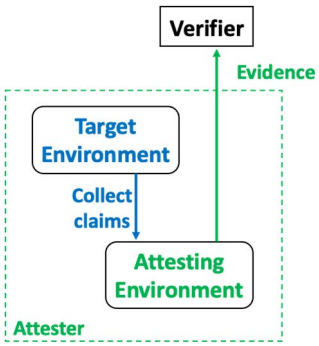


Figure 5.3.2.1-2: Structure of an attester

A verifier consumes and appraises the evidence produced by an attester by matching its attributes against collections of reference measurements. As the attester role can be realized through varying implementations, the industry has recognized the need for standard information and data models to describe evidence claims and reference measurements. This would help the different supply chain actors (e.g. hardware, software, service vendors) develop compliant solutions that scale the deployment of a zero-trust security model that utilizes attestation. The IETF Concise Reference Integrity Manifest (CoRIM) [i.18] and related efforts by the TCG [i.23] are aimed at such standardization which could be integrated into ETSI ISG MEC standards.

5.3.2.2 Assessing trustworthiness of MEC applications

Virtualized workloads such as MEC applications are deployed on a layered execution environment as depicted in figure 5.3.2.2-1. An attestation framework may be implemented to collect evidence about all critical software layers that are relevant to the security of the MEC application and convey the evidence to a verifier.

As an example, each layer of the execution environment may take on the role of an attester by compiling evidence about the next layer during a secure boot process. Evidence from each layer may also be collected at a physical or virtual RoT entity (e.g. Virtual Trusted Platform Module (vTPM) or Trusted Execution Environment (TEE)) to be sent to the Virtual Machine Monitor (VMM) or hypervisor in the virtualization infrastructure. The VMM can relay the evidence to a verifier in response to an attestation request for the MEC application. Evidence may additionally be included about the VMM's own trustworthiness and that of other remaining components along the chain of trust ending at a hardware RoT entity on the MEC host. The hardware RoT entity would in turn be validated via endorsements by an endorser (e.g. a manufacturer).

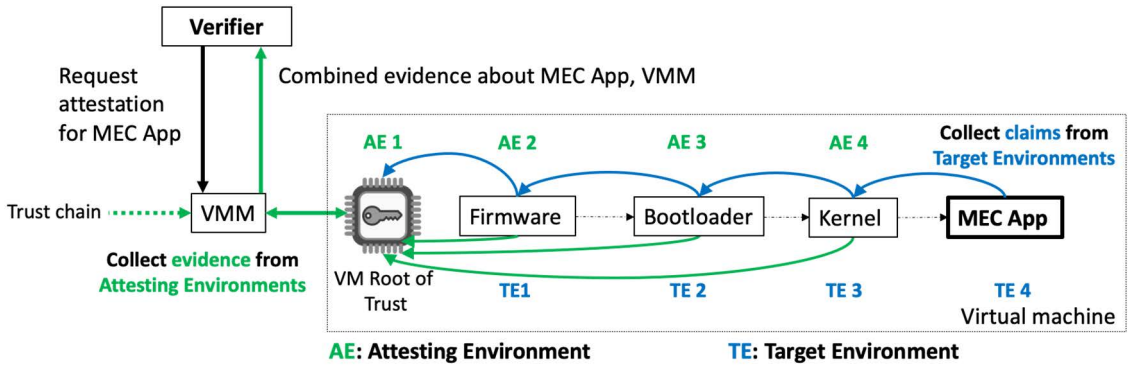


Figure 5.3.2.2-1: Collecting evidence of trustworthiness of a MEC application

NOTE: In practice, it would be up to the vendor/implementation to collect appropriate evidence about the MEC application and its hosting virtual environment. For example, the Linux OS supports secure boot functionalities and the Linux kernel's integrity subsystem [i.25] can collect claims about executed software which may be harnessed to produce evidence as illustrated in figure 5.3.2.2-1.

5.3.2.3 Attestation framework for MEC applications

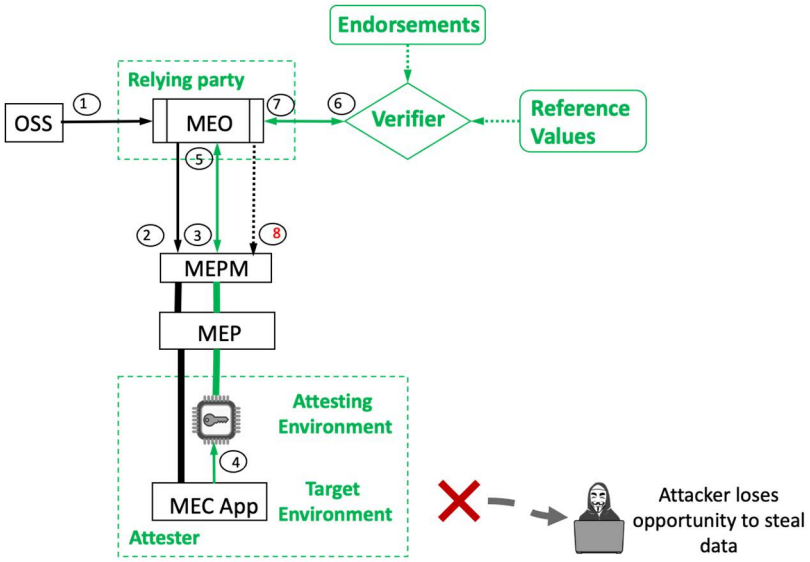


Figure 5.3.2.3-1: MEO verifying the provenance of instantiated MEC applications

Figure 5.3.2.3-1 illustrates a conceptual workflow for verifying the provenance of MEC applications in which the MEO is the relying party. The figure identifies the different participants and interactions (in **green**) for carrying out remote attestation per IETF RFC 9334 [i.8]. It assumes secure signalling pathways established between the MEO, verifier and related entities along with the ability to manage and allocate trusted computing resources to MEC applications. A MEC application is deployed with the attester role as the *target environment* that is measured by an *attesting environment* (e.g. a TEE chained to a RoT) to return evidence of its trustworthiness. This is in line with the pattern depicted in figure 5.3.2.1-2.

Trustworthiness claims are collected about the MEC application and its execution environment through a vendor/implementation defined scheme to generate evidence in response to an attestation request. Clause 5.3.2.2 presents one such scheme. The evidence appraisal process by the verifier certifies the provenance of the MEC application.

NOTE 1: Reference values for all the measured components would have to be collected prior to deployment of the MEC application to establish a trusted baseline state and the MEC system is expected to have access to these reference measurements (for e.g. it may be populated during MEC application registration).

The numbered steps in figure 5.3.2.3-1 represent the following actions:

1) The OSS instructs the MEO to instantiate a MEC application.

2) The MEO selects an appropriate MEC host to deploy the MEC application based on its requirements. The MEC application is instantiated at the selected MEC host.

3) The MEO (as the relying party) issues an attestation request for the MEC application.

4) The attesting environment bound to the MEC application collects and authenticates trustworthiness claims from the application to return as evidence.

NOTE 2: The MEC application is the target environment of the attester and thus this operation does not impose any new requirements on the MEC application itself.

5) The MEO receives evidence about the MEC application.

6) The MEO forwards the received evidence about the MEC application to the verifier to appraise the trustworthiness of the MEC application.

NOTE 3: A verifier should be run in a trusted and hardened environment, for example, in the MEC management domain.

7) The verifier appraises the evidence using reference values and endorsements and returns attestation results to the MEO.

NOTE 4: The data flows in the above steps between the MEO (the relying party), the attester and the verifier reflect the "background-check model" specified in IETF RFC 9334 [i.8], clause 5.2.

8) If the attestation results invalidate the MEC application, the MEO may suspend it. Consequently, a purported adversary would lose connectivity to the MEC application to conduct attacks.

NOTE 5: The MEO may periodically request the verifier to assess the trustworthiness of the MEC application during its lifecycle to become aware of any changes that may have compromised its runtime state. Any detected change would trigger a suspension of the MEC application.

NOTE 6: The method of interpretation of attestation results to assign a trust level and of actions to be taken in response is out of scope of the present document.

5.3.3 Solution proposal #2: Verify integrity of MEC applications through cryptographic attestation

5.3.3.1 Description

This solution proposes an example of how MEC applications integrity can be verified, before instantiation, based on IETF RFC 9334 [i.8]. This solution is complementary and may be used in conjunction with the solution described in clause 5.3.2.3 of the present document.

The MEO performs the roles of both the Verifier and the Relying Party, while the attesting environment contains the VIM, and a co-located or separate trusted hardware or privileged code to perform the measurements on the MEC app. A static measurement in the form of a securely computed application image hash is the evidence provided by the attesting environment. The evidence is compared to the MEO-stored hash which is the reference value stored during the onboarding verification. The attestation process occurs before the instantiation of the MEC application to reduce the risk of a malicious MEC application being instantiated and causing harm to the MEC system.

Table 5.3.3.1-1: IETF RATS Mapping to MEC

IETF	Role or Artifact	MEC Translation in clause 5.3.3
Relying Party	Role	MEO
Verifier	Role	MEO
Attester	Role	MEC application plus trusted hardware -- part of VIM
Evidence	Artifact	Computed application image hash
Reference Value	Artifact	Stored image hash

5.3.3.2 Solution Details

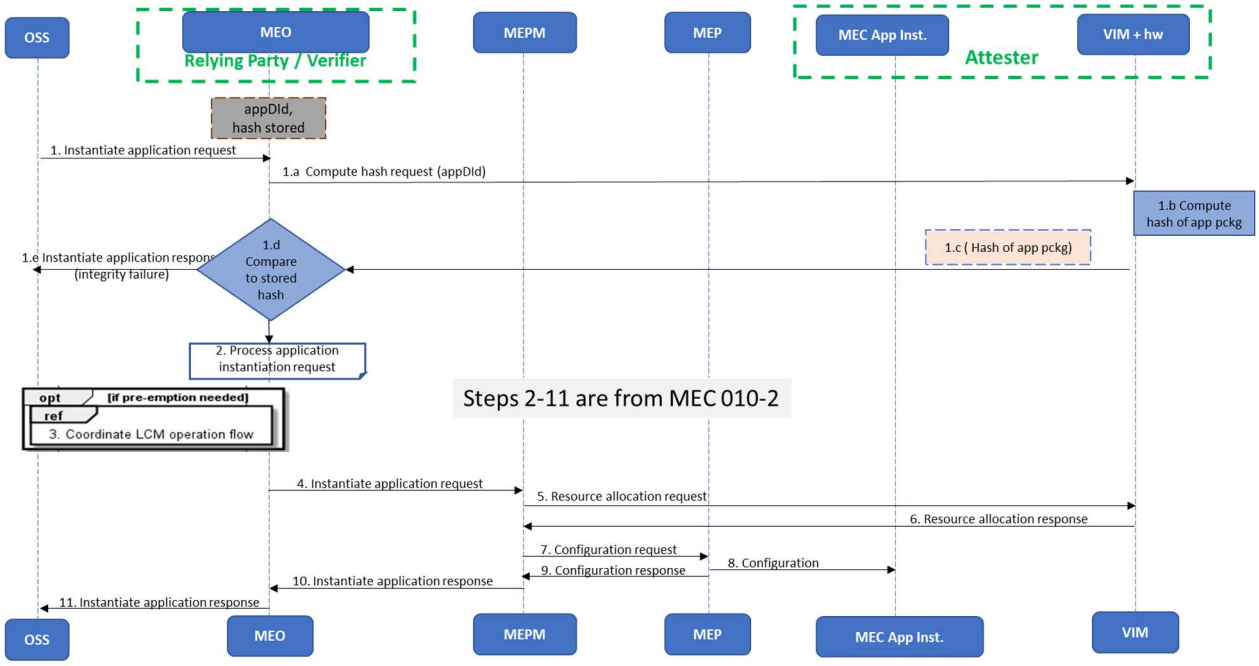


Figure 5.3.3.2-1: Attestation for MEC applications: MEO as verifier example

The numbered steps in figure 5.3.3.2-1 represent the following procedure, which is an expansion of figure 5.3.1-1 of ETSI GS MEC 010-2 [i.30]:

1) The OSS sends an instantiate application request to the MEC Orchestrator.

Proposed Attestation Process:

1.a) The MEO initiates a Compute hash request towards the VIM over Mm4 interface. This message contains the identifier of the application package files to use; the VIM has access to the downloaded application package.

1.b) The VIM computes a hash of the application package.

1.c) The VIM sends the application package hash to the MEO.

1.d) The MEO compares the application package hash from VIM (evidence) to stored known good hash (reference value).

1.e. In case there is no match, the attestation is considered to have failed and so the MEO responds to the OSS with the message Instantiate Application Response (new error code such as "integrity failure").

Resume Application Instantiation if hash comparison was successful:

The rest of the steps of the figure (steps 2 to 11) are from figure 5.3.1-1 of ETSI GS MEC 010-2 [i.30]:

2) Process application instantiation request

3) Coordinate LCM operation flow (Optional)

4) Instantiate application request

5) Resource allocation request

6) Resource allocation response

7) Configuration request

8) Configuration

9) Configuration response

10) Instantiate application response

11) Instantiate application response

5.3.4 Solution proposal #3: Continuous verification of integrity of MEC applications through periodic attestation

5.3.4.1 Description

The purpose of periodic attestation is to detect any compromise (e.g. unauthorized change) to applications while they are running and take action to protect the MEC system. Periodic attestation should be done based on a policy, which the MEC system can set forth.

Policy parameters for periodic attestation may include:

a) entity responsible for setting the policy (e.g. OSS), and where it is configured/stored;

b) whether it is done per application package (assuming it is in IN_USE) or just per application instance;

c) frequency of attestation requests;

d) action to take in case of a failed attestation; and others.

5.3.4.2 Solution Details

This solution aims to formalize the notion of periodic attestations (see note 5 of Solution #1) and is compatible with both Solutions #1 and #2. The periodic attestation flow for running application instances shares the steps (3 to 7) with the initial attestation flow described in clause 5.3.2.3. The steps preceding and following serve to enforce a regimen of conducting periodic attestations of MEC applications based on set policy.

The following procedure may apply to a single application instance or multiple application instances depending on periodic attestation policy:

0) The OSS or other entity instructs the MEO to perform periodic attestation, per policy.

1) The MEO initiates the periodic attestation process for application instance(s) according to the policy parameters.

2) The following is performed:

Steps 3 to 7 of solution 5.3.2.3.

8) If the attestation process fails for a given MEC application instance, the MEO should immediately terminate it.

9) Optionally, the MEO may convey the application instance(s) attestation results to the OSS (failure/success).

5.3.5 Evaluation

Solution #1 in clause 5.3.2 outlines a comprehensive framework with which a MEC application instance may be evaluated for its parity to a trusted baseline state. Any unauthorized change that occurs following the deployment of the MEC application would result in a mismatch of its component measurements (collected as part of attestation procedures) with their reference values and termination of the MEC application instance. This helps to deter attacks characterized by the threat described in clause 5.3.1.

Solution #2 in clause 5.3.3 helps to secure MEC application instantiation through cryptographic attestation. A static hash measurement occurs before instantiation to verify the integrity of the application package image hash and by extension, the application instance. This solution ensures the integrity of the application instance during the instantiation process.

Solution #3 in clause 5.3.4 helps to reassert the security of *running* MEC application instances through periodic attestations. This allows for regular and frequent MEC application verifications per a specified policy, enabling the constant monitoring and detection of any signs of potential unauthorized modifications or tampering performed during operation.

5.4 Key issue #4: Compromise of application package during on-boarding

5.4.1 Description

The process of lifecycle management (LCM) of applications running on a MEC host presents opportunities for threat actors to compromise the application.

The LCM is the responsibility of the MEO. According to ETSI GS MEC 010-2 [i.30], the MEO runs two types of application lifecycle management: a) application package management and b) application instance management.

The application package management consists of the following flows: on-boarding, life management (fetch, query, enable, disable), and deletion. A MEC application during each of these flows, and even during the application development stage preceding these, may be subject to various threats.

To defend against these threats, it is necessary to verify that an application package is trustworthy and safe to deploy. The thoroughness of the checks performed during this verification process can vary depending on the security requirements of the actual MEC system, and might include various assessment steps such as sandbox testing.

Such checks can for instance be done by an entity in the OSS. The OSS then re-signs the package to confirm its trustworthiness and safety after passing the needed tests. In such case, the trust model is that the OSS trusts the application package provider/vendor and the MEO trusts the OSS.

The following security threats are possible in the context of this key issue:

- If secure development processes are not employed, an insider or malicious actor may get access to the MEC application vendor development process and insert malicious code within a MEC application package.
- If a MEC application package is not integrity protected, a malicious actor could tamper with the data contained in the package before it is onboarded by the MEO.
- If the MEC application package trustworthiness and safety is not verified, then a potentially malicious application may be sent to the MEO for on-boarding.
- If the MEO does not verify all incoming and locally stored MEC application packages from the OSS, the MEO may unknowingly onboard malicious application packages.

The following recommendations pertaining to application package protection are proposed to mitigate the threats outlined above:

- The application package should be signed by the application provider/vendor.
- The OSS should verify the application package signature by the application provider. If a signature is present and verification fails, the application package onboarding process should halt.
- It should be verified that the application package meets trustworthiness requirements, and if so, the OSS should sign the application package before sending it to the MEO.
- The MEO should verify both the application package signatures, the one by the provider and the one by the OSS. If either of these signature verifications fails, then the onboarding process should halt.

5.4.2 Solution proposal #1: Secure onboarding of MEC application packages

5.4.2.1 Description

The OSS and MEO are the two entities that handle the MEC application package during the onboarding lifecycle phase. According to clause 5.2.2 (On-board application package) of ETSI GS MEC 010-2 [i.30], the OSS initiates the MEC onboarding process by sending a package request to the MEO with the application package included. The MEO receives and processes the request, which includes validating the authenticity and integrity of the application package. The MEO sends back an acknowledgement to the OSS if the onboarding is successful, and the application package is both enabled and available for deployment in the MEC system.

While ETSI GS MEC 010-2 [i.30] calls for the signing of the application package by the application provider and the authenticity/integrity check by the MEO in clause 5.2.2, some threats remain as outlined in clause 5.4.1 of the present document. To that end, this solution proposes to extend the role of the OSS to be the entity that performs application trustworthiness checks for the MEC system, beyond signature verification.

To guard against the on-boarding threats, there should be a way to show explicit OSS authorization for application onboarding (and consequently, deployment). This way, the MEO can be assured that the OSS itself vouches for the integrity, authenticity and trustworthiness of the application package. Thus, this solution proposes to strengthen the security of the MEC application on-boarding process through a dual-signature mechanism involving both the OSS and the MEO.

5.4.2.2 Solution details

The solution proposal involves adding an application verification step and one additional signing step at the OSS before the on-board application package request is sent from the OSS to the MEO, and one additional verification step at the MEO. The OSS verifies the authenticity, integrity and trustworthiness of the application package using at least the supplied application provider signature and other mechanisms/tools. If these checks are successful, the OSS will re-sign (meaning adding its own brand-new signature to) the application package.

The OSS verification and re-signing of the application package demonstrates authorization for the MEC application deployment. The OSS will then send the signed package to the MEO in the application package request as part of On-board application package request. The MEO will then use the signatures from both the application provider and the OSS to perform the validity check for the application package. More details are depicted in figure 5.4.2.2-1.

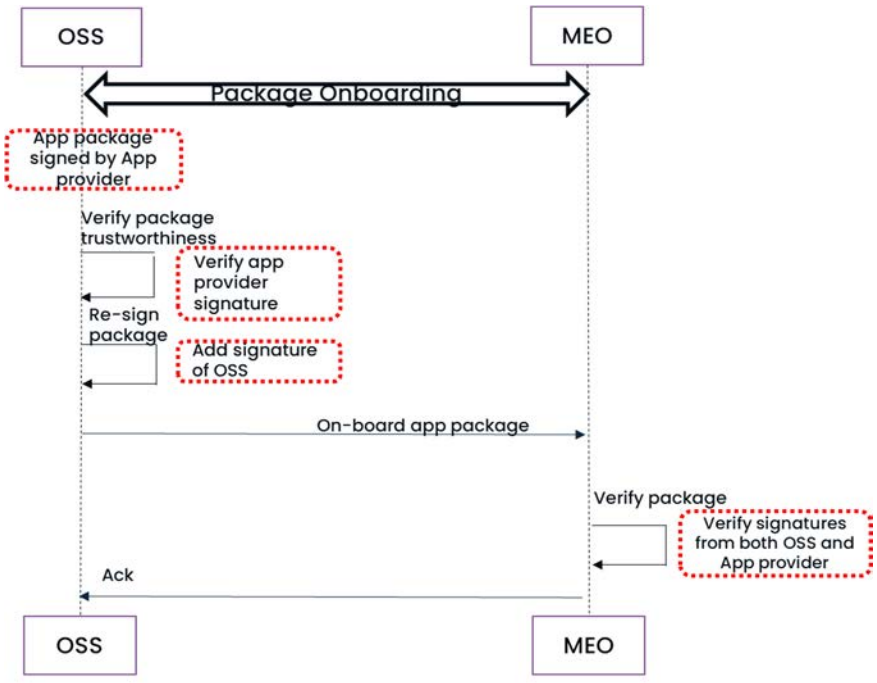


Figure 5.4.2.2-1: Secure MEC App package on-boarding process

An updated procedure for clause 5.2.2 (On-board application package) of ETSI GS MEC 010-2 [i.30] is described below:

1) (*proposed text*): The OSS checks the authenticity, integrity and trustworthiness of application package using at least the application provider signature, and if successful, re-signs (adds its own signature to) the package.

2) *(existing text*): The OSS sends an on-board application package request to the MEC Orchestrator, in which, the MEC application package is included.

   The MEC Orchestrator checks the application package, for example the MEC Orchestrator checks for the existence of mandatory elements within the application package, validates the authenticity and integrity of the application package.

   (proposed text): The MEC Orchestrator uses the signatures from both the application provider and OSS to perform the application package validation;

   (existing text): and the MEC Orchestrator checks the format of the application image and format of the application rules and requirements.

   The MEC Orchestrator allocates a unique application package ID for the on-boarded MEC application package and related status information, and keeps a record of on-boarded application packages. Optionally the MEC Orchestrator prepares the virtualisation infrastructure manager(s) with the application image (e.g. by sending the application image to appropriate virtualisation infrastructure managers). Alternatively, such preparation may be done later, but needs to be finished before the application is instantiated. The MEC Orchestrator notifies the subscribers to AppPackageOnBoardingNotification of the on-boarding of the MEC application package.

3) The MEC Orchestrator acknowledges the application package on-boarding to the OSS. The application package is now available in the MEC system and enabled.

5.4.3 Evaluation

The proposed solution helps increase the security posture of the MEC system during MEC application onboarding through the introduction of a dual-signature process involving verification at both the OSS and the MEO.

The solution enables the entities handling the application package to ensure its authenticity, integrity and trustworthiness, and introduces an additional step to the existing application package validity check in clause 5.2.2 of ETSI GS MEC 010-2 [i.30], On-board application package: the OSS performs additional checks and conveys the result to the MEO.

The proposed solution procedure ensures safe and authorized application deployment, such that any malicious or unauthorized application packages or modifications to the application package may be discovered and mitigated in the MEC application package onboarding process.

5.5 Key issue #5: Compromise of application during updates

5.5.1 Description

MEC Application(s) may need to be updated during operation for security or modification of features. Ultimately, it is up to the MEC operator's discretion as to whether or not to update a MEC application, but updates pertaining to security should be prioritized and be applied in a timely manner. Doing so allows a potentially susceptible application, and by extension, the MEC system its deployed on, to be secured from its currently known vulnerabilities. This update process may be subject to threats, which this KI attempts to highlight.

According to ETSI GS MEC 010-2 [i.30], the MEO runs two types of application lifecycle management:

a) Application package management; and

b) Application instance lifecycle management.

Since the MEC applications may be deployed either as Virtual Machines (VMs) or containers, the update process of an application *may* consist of *multiple* message flow sets as described in ETSI GS MEC 010-2: termination and deletion of the now outdated version, and onboarding and instantiation of the new current version. Note that the MEO, which initiates these operations, may not be aware that they are part of the application update process.

Presented below is a combination of message flows from ETSI GS MEC 010-2 [i.30] that *may occur* during the update process:

Table 5.5.1-1: Applicable message flows during MEC application updates

Name of Message Flow	New Version of Application	Older Version of Application
5.2.2 On-board application package	X	
5.2.5 Enable application package	X	
5.3.1 Application instantiation	X	
5.3.3 Application operation (start/stop)	X	X
5.3.2 Application termination		X
5.2.4 Disable application package		X
5.2.6 Delete application package		X

The following security threats are possible in the context of this key issue:

- If application updates are not done securely, the application update packages may be tampered with or corrupted during onboarding.
- If the target application version is not verified, an unintended and/or potentially vulnerable version of the application may be deployed.
- If applications are not updated with their latest security updates, the application will be susceptible to current vulnerabilities.

NOTE: Threats applicable to onboarding and termination of a MEC application are also applicable to MEC application updates.

5.5.2 Solution proposal #1: Secure MEC Application Update

5.5.2.1 Description

MEC application update is a process whereby a MEC application is updated for any reason (e.g. security, feature modifications, etc.). It involves operations from both the application lifecycle management processes:

a) Application package management; and

b) Application instance lifecycle management.

The OSS oversees the overall process and all the procedures involved.

The MEC application update process can be categorized into two different message flow sets: Deletion of the application package with the older version with termination of all current application instances, and onboarding of the application package of the newest version along with instantiation of application instances from the newest version.

Securing the MEC application update can be achieved by combining security measures for the applicable MEC application message flows. The message flows that are part of application updates are found in ETSI GS MEC 010-2 [i.30].

5.5.2.2 Solution details

The following security recommendations are made.

- For the outdated application package deletion, the solution in clause 5.6.2 is applicable.
- For the onboarding of the newest version of the application package, the solution in clause 5.4.2 is applicable.

- For the instantiation of an application instance from the newest version of the application package, the solutions under clause 5.3 are applicable.

5.5.3 Evaluation

The proposed solution in clause 5.5.2 addresses the threats outlined in clause 5.5.1. In particular, the proposed solution helps mitigate the applicable threats in the related message flows of ETSI GS MEC 010-2 [i.30] during application updates, namely:

1) 5.2.2 On-board application package;

2) 5.3.1 Application instantiation; and

3) 5.2.6 Delete application package.

Executing these message flows with the proposed security solutions provides an overall secure approach for updating MEC applications.

5.6 Key issue #6: Threats associated with Application Package Deletion

5.6.1 Description

According to ETSI GS MEC 010-2 [i.30], the MEO runs two types of application lifecycle management:

a) Application package management; and

b) Application instance lifecycle management.

Application package deletion involves operations from both of these lifecycle management processes.

The reasons for application package deletion can be due to OSS/BSS processes, or as part of application updates for security reasons (patching) or for feature enhancements. If a vulnerability is found in an application instance or package, it is advisable to terminate all the application instances and remove the corresponding application package from the MEC system in a timely manner; this is to reduce the risk that adversaries will exploit it and cause loss of confidentiality, integrity and/or availability of affected MEC system components.

Vulnerabilities are usually found in code - e.g. an adversary finds a programming mistake that they can exploit in the context of a running application instance. Therefore, if such a vulnerability is discovered, all the application instances of the affected application package should be terminated and the package itself be deleted in a timely fashion.

NOTE 1: The OSS may set an alarm upon discovery of a vulnerability. After the actions to handle this threat are completed, the OSS should clear the alarm.

As MEC applications may be deployed either as Virtual Machines (VMs) or containers, the process of application package deletion may consist of several separate steps: application instance termination, disabling of application package, and deletion of application package. The operations that are part of application package management and application instance lifecycle management and which pertain to application package deletion are shown below from ETSI GS MEC 010-2 [i.30]:

Table 5.6.1-1: Message flows associated with application package deletion

Name of Message Flow	Process	Pre-Condition(s)	Brief description	Post-Condition(s)
5.2.4 Disable application package	App Package Management	Application package is ONBOARDED and ENABLED.	Marks an application operational state as "DISABLED" in the MEC system.	New application instances cannot be instantiated. Application is now in operational state DISABLED, but the usage state may be "IN_USE" or "NOT_IN_USE" (clause 6.3.3.7.1 of [i.30]).
5.2.6 Delete application package	App Package Management	Application package is DISABLED, and NOT_IN-USE.	Removes an application package from the MEC system. The application package is marked as DISABLED and NOT_IN_USE in order for this operation to succeed. (clause 6.3.3.9.1 of [i.30]).	Application package does not exist in the MEC system; application itself cannot be instantiated.
5.3.2 Application termination	App instance LCM	Application is instantiated (at least one instance of the application exists).	Terminates a given application instance regardless of its operational state. This procedure involves resource deletion as well.	The application instance no longer exists.

The OSS initiates the process of application package deletion. The MEO oversees the message flows listed in table 5.6.1-1 separately.

NOTE 2: The MEO may not be aware that these message flows are part of an application package deletion process for purposes of threat handling.

For a given application package, if more than one application instance is currently running, then all of these instances have to be terminated in order for the OSS to be able to delete the application package (clauses 5.2.6, 5.3.2 and 6.3.3.9.1 of ETSI GS MEC 010-2 [i.30]). However, the disabling of an application package can be followed by one or more application instance termination message flows (since other application instances may still exist: the application package can be DISABLED, but IN_USE - table 6.2.3.3.2-1 of [i.30]).

The following security threats are possible in the context of this key issue:

- After a vulnerability is found in an application instance or an application package, if the application package is not deleted in a timely fashion, adversaries may exploit the vulnerability and cause harm to the MEC system.
- Any cryptographic or privacy-sensitive data may be recoverable from the memory resources used by the terminated application instance.
- The resources may be improperly released during application instance termination, resulting in other application instances unable to make use of available resources.

5.6.2 Solution proposal #1: Secure application package deletion

5.6.2.1 Description

Application package deletion is a process whereby an application ceases to exist in the MEC system, and it involves operations from both the application lifecycle management processes:

a) Application package management; and

b) Application instance lifecycle management.

The OSS oversees the overall process and all the procedures involved.

Application package management aspects: the OSS which oversees the application package deletion needs to properly document this event: e.g. date/time of application package deletion, the AppPkgInfo and other such identifying application package data, optionally reason for deletion. Such data may be provided by the OSS to a Business Support System (BSS) component if one is supported by the MEC operator, for example to show timely response to the discovery of vulnerabilities.

Application instance lifecycle management aspects: in order to secure the application package deletion processes, all application instances of this application package need to be terminated, and the Virtualization Infrastructure Manager (VIM) needs to properly release resources from the terminated application instances; in addition, the VIM needs to ensure that no secret data is left in memory after resource release, risking recovery by a malicious application with access to the same resources.

5.6.2.2 Solution details

The following security recommendations are made:

- For each application instance termination procedure: upon receiving a Resource deletion request, the VIM should clear (e.g. "zeroize" or repeatedly overwrite) all cryptographic or privacy-sensitive data that has been used by the terminated application instance, and then send the Resource deletion response.
- For each application instance termination procedure: upon receiving a Resource deletion request, the VIM should ensure that all compute, storage and network resources used by that instance have been completely released. The VIM may then send the Resource deletion response.
- For each application package deletion: the OSS should document in a secure fashion the application package deletion event.

5.6.3 Evaluation

The proposed solution addresses the threats outlined in clause 5.6.1 and provides a more secure process for application package deletion. It ensures that sensitive data are securely removed, and resources consumed by application instances during operations are properly released during application instance termination(s) for the deleted application package. Documentation of the application package deletion event is also specified for purposes of auditability and analysis.

5.7 Key issue #7: MEC App anomalous behaviour

5.7.1 Description

The MEC system is likely to become a target of cyber-attacks. The ability to detect anomalous behaviour of MEC applications is a key component in thwarting adversary actions; this task can be accomplished by monitoring for anomalous behaviours and indicators of potential compromise. Behaviours that can be monitored to detect compromise include excessive resource usage, unusual traffic patterns, unauthorized data access, connection attempts to grey-listed URLs, etc. Security Monitoring and Management (SMM) allows the collection of these metrics to observe potentially malicious behaviour and provides the facility to take action to mitigate their causes.

As an example, consider a MEC system subject to a Distributed Denial of Service (DDoS) attack initiated by an IoT botnet. Each IoT device may request access to a resource associated with an instantiated MEC app. If the MEC app can access the Internet, the resource in question may be TCP connections, for example. The MEC host, via its Virtual Infrastructure (VI) implementation, will possess a finite number of such resources (e.g. compute, networking, or storage). In the case where there is no maximum resource usage defined and enforced for the MEC app instance to consume, eventually the DDoS attack on one MEC app will utilize sufficient resources that other MEC apps will be impacted (i.e. experience limited access to that resource). As the attack continues to scale up, the finite set of resources will become devoted to that one MEC app, potentially causing a general loss of MEC service by that MEC host.

Detection of cyber-attacks is facilitated by monitoring MEC application behaviour, consequently increasing the likelihood of effective response before an adversary inflicts significant damage. MEC Application anomalous behaviours that may be detected include:

- Unusual access to other MEC entities (e.g. MEC platform).

- Unusual access to other MEC Apps and data.
- Unusual access to malicious external sites.
- Unusual communication or resource consumption that could indicate the presence of persistent adversary activity in the MEC system.

NOTE: The issue of targeted cyber-attacks and resulting anomalous behaviour applies to the MEC architectural entities and is not constrained just to MEC applications.

5.7.2 Solution proposal #1: Security Monitoring and Management for MEC

5.7.2.1 Description

A Security Monitoring and Management (SMM) system may be used to identify compromised MEC entities and mitigate the damage they may incur to the MEC system. An SMM system employs behavioural monitoring to identify MEC entities that are taking actions that are outside expectations for proper and secure operation for the identified entity. Once identified, an SMM system, through its management component, may take steps to limit the operation (and hence potential incurred damage) of the identified entity. For example, if the identified MEC entity is a MEC app behaving in an anomalous manner, the MEC app may be terminated.

Security monitoring and management for NFV systems is specified in ETSI GS NFV-SEC 013 [i.31] and ETSI GS NFV-SEC 024 [i.36]. Aspects of the NFV-SEC specified SMM may be re-used by MEC (where applicable). However, some customizations and augmentations are needed to define an SMM for MEC systems.

The target for NFV-SEC specified SMM is Virtual Network Functions (VNFs) and, more broadly, the Network Services (NSs) they comprise. VNFs run on Virtual Infrastructure (VI) within a multi-tenant NFV system that may include defined trust domains. These aspects may not be directly applicable to defining a MEC SMM. MEC systems host MEC apps via VNFs and NSs. MEC systems are not explicitly multi-tenant and trust domains are not currently specified for MEC. However, the functionality of the NFV-SEC specified SMM, as exemplified in the following SMM System Functional Architecture (SMM FA) diagram, may be re-used by MEC.

Referring to figure 5.7.2.1-1, an SMM system interfacing to a MEC system could include the following Functional Elements (FEs):

- Security Monitor
- Data Store
- Analytics Engine
- Security Manager
- Policy Store
- Security Profile Store

The interfaces between the MEC system and the SMM system (indicated as Ms interfaces) are expected to be standardized within the MEC specifications. These interfaces are for collection of security monitoring related data from MEC entities and for sending security directives from the Security Manager (as necessary to mitigate attacks) to these MEC entities. Also, the OSS may use these interfaces to control the SMM and be informed of its operation.

The FEs in figure 5.7.2.1-1 are described. The Security Monitor collects security-related data from MEC entities and deposits it into the Data Store. The Analytics Engine FE uses data from the Data Store to check and potentially flag MEC entity anomalous behaviour. The Security Manager generates and dispatches security directives to MEC entities based upon attacks that have been detected by the Analytics Engine FE. For example, a security directive may instruct a MEC entity under attack to curtail or terminate indicated operations. The Security Profile Store stores the security profiles of the MEC entities in the MEC system. The Security Manager also coordinates SMM operation according to applicable policies from the Policy Store and knowledge of the currently running MEC system entities stored in the Security Profile Store. The Security Manager directs the Security Monitor what data to collect from which MEC entities at what frequency.

An important feature built into NFV-SEC SMM operation is flexibility as implemented by *personalization and policy provisioning* (clause 7.6.5 of ETSI GS NFV-SEC 013 [i.35]) which can be exemplified in a MEC context. The data that the SMM system collects from MEC entities, the behaviour of MEC entities that may subsequently raise a security red flag, and the management action(s) taken when a MEC entity misbehaves are subject to definition/enforcement within the MEC system via security policies and security profiles. Executing a particular security policy (policies), the SMM system will detect specific MEC entity aberrant behaviour and take specified actions if found. The SMM security policy currently in effect is chosen by the MEC system operator to best balance secure and efficient operation of the MEC apps within the MEC system.

Each MEC entity is personalized by a defined security profile. This profile indicates what security-related data a MEC entity can deliver to the SMM system as well as what security directives it will respond to and how. By collecting the security profiles for the MEC entities operating within a MEC system, the SMM system stores within its Security Profile Store what data it can collect and what effect it can have on MEC system entities. Security policies and security profiles may evolve over time as security approaches are refined and new MEC application types are fielded. By utilizing an SMM system, security-related attacks may be detected and mitigated in a uniform, scalable manner.

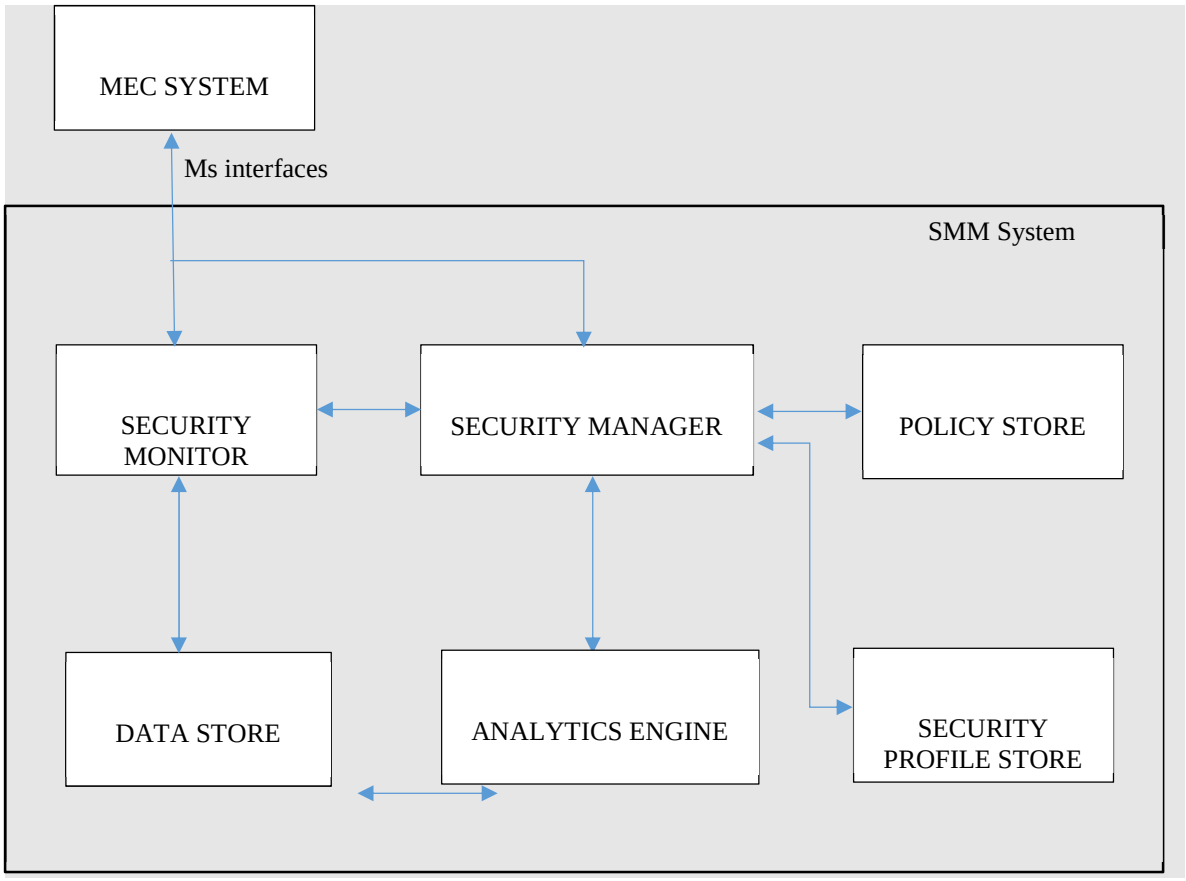


Figure 5.7.2.1-1: MEC System interfaced to an SMM System Functional Architecture

Regarding the example use case of a DDoS attack initiated by an IoT botnet in clause 5.7.1, a simple solution is to enforce equitable sharing of the resource across MEC apps within a MEC host. In this case, each MEC app would be allocated a resource amount (e.g. M GIPS, N connections or R GB). The SMM system would monitor the MEC app's usage of that resource. During the described DDoS attack, the SMM may become aware of the increased use of the resource approaching the total available allocation for that MEC app; a security-related suspicious behaviour would be detected, and the Security Manager may send a message to the MEC app to take action to mitigate the attack. Such action could be to throttle back or refuse all further client resource requests for that MEC app. If the attack is not mitigated, the Security Manager could request the MEO to terminate the MEC app under attack.

Note that the security profile of the MEC app and the MEO, as well as the appropriate SMM security policy are necessary elements to support this use case. Specifically, the MEC app security profile indicates availability of the resource usage count to the Security Monitor. The security profile of the MEC app and MEO indicates that the MEC app will accept (and will respond to) the indicated mitigation messages. The indicated monitoring and management actions are expected to be consistent with the SMM policy currently in effect.

5.7.2.2 Solution details

The following security recommendations are made:

- Enable SMM to monitor and mitigate potential MEC system compromises and attacks.

5.7.3 Evaluation

The solution proposes a mechanism for ETSI MEC to enable Security Monitoring and Management (SMM) of MEC applications and MEC architectural entities. Operators may customize the SMM system configuration to fit their unique context and needs. Anomalous behaviour potentially indicative of a security event can be detected and mitigated by the SMM system. Interfaces connecting the MEC system to the SMM system enable collection of security-related data and message exchange that can allow identification, mitigation, and management of potential security incidents in the MEC system.

ETSI ISG MEC may consider the following recommendation:

- Create interfaces enabling SMM system to be utilized in the MEC systems.
- Specify, if necessary, the type of data to be collected via these interfaces.

6 Gap analysis and recommendations

Table 6-1 summarizes all the key issues identified in clause 5 along with their remedial solution(s). It also highlights the related technical or architectural gaps in the existing specifications as potential targets for future normative work.

Table 6-1: Summary of key issues and solutions

Key issue	Gap	Solution	Potential specification Impact
Stolen MEC App access tokens	MEC App authentication/authorization uses OAuth 2.0 access tokens that are subject to this threat	mTLS client authentication with client certificate-bound access tokens	ETSI GS MEC 009 [i.12]: Patterns for OAuth 2.0 authorization using IETF RFC 8705 [i.15] as per clause 5.1.2.
Stolen MEC App identity	No formal assurances of the protection of the MEC App credential private key	Enhance MEC App authentication using hardware-based security and attestation procedures to establish trust in the protection of MEC App credentials	ETSI GS MEC 003 [i.11]: Propose an optional RoT entity and describe its relationship to a MEC App instance. ETSI GS MEC 010-2 [i.30]: Propose optional use of RoT entity as part of applicable message flows. ETSI GS MEC 011 [i.40]: Define an optional API for acquiring credentials protected by a RoT entity using attestation procedures based on IETF RFC 9334 [i.8] and as per clause 5.2.2.2.

Key issue	Gap	Solution	Potential specification Impact
Compromised MEC applications, asset theft	No mechanism to securely assess trustworthiness of MEC Apps with respect to an established baseline state	Verify provenance of MEC applications through cryptographic attestations	ETSI GS MEC 003 [i.11]: Propose an optional RoT entity and describe its relationship to a MEC App instance. ETSI GS MEC 010-2 [i.30]: Propose optional use of RoT entity as part of applicable message flows. ETSI GS MEC 011 [i.40]: Define an optional API to retrieve evidence of trustworthiness of MEC application instances using attestation procedures based on IETF RFC 9334 [i.8] and as per the solutions in clause 5.3.
		Continuous verification of integrity of MEC applications through periodic attestation	
		Verify integrity of MEC applications through cryptographic attestation	
MEC app anomalous behaviour	No mechanism to detect anomalous behaviours and potential indicators of compromise for the MEC system	Security Monitoring and Management for MEC	ETSI GS MEC 002 [i.41]: Add optional requirement(s) for security monitoring and management. ETSI GS MEC 003 [i.11]: Define interfaces enabling SMM system to be utilized by the MEC system as per clause 5.7.3. Specify in a new SMM API spec the type of data to be collected by SMM system as per clause 5.7.3.
Compromise of application during updates	Lack of security guidance for updating a MEC application.	Secure MEC Application Update	ETSI GS MEC 010-2 [i.30]: Update applicable message flows to enable secure updates to MEC applications as per clause 5.5.3.
Compromise of application package during on-boarding	No mechanism for MEO to verify application package as trustworthy to deploy onto MEC system	Secure onboarding of MEC application packages	ETSI GS MEC 010-2 [i.30]: Add validity check to MEC application package onboarding procedure as per clause 5.4.3.
Threats associated with Application Package Deletion	No assurance of securely deleting application package	Secure application package deletion	ETSI GS MEC 010-2 [i.30]: Add recommendations as per clause 5.6.3.

ETSI ISG MEC may take the above recommendations into further consideration.

Annex A:
Change History

Date	Version	Information about changes
August 2021	V3.0.1	Scope and initial skeleton of the study
December 2021	V3.0.2	Initial content for sections 2.x-4.x
November 2022	V3.0.3	Updated structure with key issues/solutions, new contents for clauses 4.6 and 5.x
January 2023	V3.0.4	New contents for clauses 5.3.2 and 5.4
March 2023	V3.0.5	New key issues #5 (clause 5.5.x) and #6 (5.6.x)
April 2023	V3.0.6	New contents for key issues #5 and #6, new solution for key issue #3 (clause 5.3.3)
September 2023	V3.0.7	New 4.x clause (4.5), New key issue #7, text for evaluation clauses 5.5.3/5.6.3, new solution 5.3.4
November 2023	V3.0.8	Updated references and abbreviations; text for evaluation clauses 5.2.3, 5.3.5; minor edits to clauses 4.4, 5.2.2, 5.3.2.1
December 2023	V3.0.9	Text for clause 6
January 2024	V3.0.10	Stable draft
January 2024	V3.0.11	Resolved editor's notes
January 2024	V3.0.12	Final draft V3.0.12 is similar to Stable draft V3.0.11 and ready for the MEC Remote Consensus for review
February 2024	V3.0.13	Final draft V3.0.13 with editorial changes recommended by ETSI staff

History

Document history		
V3.1.1	March 2024	Publication