AMWA NMOS API Security

Simon Rankine – Research Engineer
BBC Research and Development

Motivation

• To get the most out of the NMOS APIs we need to be able to run them on networks which we cannot fully control
• In particularly sensitive environments it may not be enough to isolate the broadcast network
• Attacks on broadcasters such as TV5MONDE highlight that the broadcast industry is a high profile target.
• Security protects against accidental as well as deliberate misuse
AMWA NMOS APIs and Security

- Part of the motivation for using HTTP is that we can harness carefully scrutinised and well developed technology from the web industry.
- As such have always been “securable” using standard mechanisms, but to do so would have broken interoperability.
- The AMWA set up the API security workgroup to investigate how the APIs could be secured in an interoperable way.

Objectives

**Confidentiality** Data passing between client and the APIs is unreadable to third parties.

**Identification** The client can check whether the API it is interacting with is owned by a trusted party.

**Integrity** It must be clear if data travelling to or from the API been tampered with.

**Authentication** The client can check if packets actually came from the API it is interacting with, and vice versa.

**Authorisation** The API can determine whether the client interacting with it has authorisation to carry out the operation requested.
Work Areas

**Connection Security**
- HTTP over TLS (HTTPS)
- Identify cipher suites for interoperability
- Establish best practice for use of TLS with AMWA NMOS APIs

**Establishing Trust**
- Public key infrastructure with x509 certificates
- Explore how PKI can be used in a broadcast environment

**Client Authorisation**
- OAuth 2.0 with JSON Web Tokens
- Identify what is needed to ensure interoperability

---

Scope

Riedel will cover this at 15:30 ->

- **Application**
- **Transport**
- **Network**
- **Physical**

- API Client Authorisation (Web Tokens)
- Session Security (TLS)
- Network access control
- Site/Server room access etc.
Connection Security

- Tunnels insecure traffic like HTTP through an encrypted connection so that it cannot be read or modified during transit.
- The TLS protocol is widely used for securing a wide range of traffic across the internet and on private/corporate networks.
- This is achieved using a collection of different algorithms, which together form the “cipher suite”.

TLS Handshake

1. Client Hello
   - Cipher suite support list
   - Protocol version

2. Server Hello
   - Protocol version
   - Server certificate
   - Cipher suites
   - Server random nonce

3. Client Key Exchange
   - Encrypted secret key

4. Change Cipher Spec
   - Client asks server to switch to the given cipher

5. Change Cipher Spec
   - Finished

6. Secure Communication commence
Cipher Suites

**TLS** ECDHE **RSA** WITH **AES** _256_ **GCM** **SHA384**

<table>
<thead>
<tr>
<th>Protocol</th>
<th>Key exchange cipher</th>
<th>Key authentication algorithm</th>
<th>Block cipher</th>
<th>Block cipher mode</th>
<th>Hashing Algorithm</th>
</tr>
</thead>
</table>
| **HTTP over TLS (HTTPS)** – Cipher Suites

- TLS ECDHE ECDSA WITH AES 128 GCM SHA256
- TLS ECDHE ECDSA WITH AES 256 GCM SHA384
- TLS ECDHE ECDSA WITH AES 128 CBC SHA256
- TLS ECDHE ECDSA WITH AES 256 CBC SHA384
- TLS ECDHE RSA WITH AES 128 GCM SHA256
- TLS ECDHE RSA WITH AES 256 GCM SHA384
- TLS DHE RSA WITH AES 128 GCM SHA256
- TLS DHE RSA WITH AES 256 GCM SHA384
- TLS ECDHE RSA WITH AES 128 CBC SHA256
- TLS ECDHE RSA WITH AES 256 CBC SHA384
- TLS DHE RSA WITH AES 128 CBC SHA256
- TLS DHE RSA WITH AES 256 CBC SHA256
- TLS ECDHE ECDSA WITH AES 128 CCM 8
**But isn’t this SSL?**

SSL 1.0, 2.0, 3.0 all insecure – don’t use them!

**TLS 1.0** No longer considered secure, avoid use.

**TLS 1.1** Still considered secure, but generally TLS 1.2 is preferred as there is little difference.

**TLS 1.2** Current best practice – use wherever possible.

**TLS 1.3** Very recently published as RFC, still finding its way into implementations.

---

**Establishing Trust**

- API servers need to hold a certificate trusted by the client. This certificate must match its subject name (e.g. URL).

- Certificates are issued by a trusted 3rd party – the “Certificate Authority”.

- Asymmetric encryption is used to allow the certificate authority to “sign” the server’s certificate such that the client can check its authenticity using the public key of the certificate authority.
From IP Showcase Theatre at IBC 2018

Curated by the Video Services Forum vsf.tv