Proposal for Adoption into the MOS Protocol

MOS Encryption and Security via Web Sockets & MOS Passive Mode

July 2017 / April 2018 – Revision 1.3

Shawn Snider
VP | Production Workflow & Cloud Operations
Ross Video Limited
ssnider@rossvideo.com

Background

As broadcast systems transition into an IT-centric world, customers are demanding a greater level of security, and heavily scrutinizing the insecure nature in which our products communicate over MOS today. Given MOS has no encryption or real-security (some devices/newsrooms do IP-based filtering only, which can be spoofed), this is a real-problem we are facing today. As well, as more of us shifting systems into cloud (private and public) based infrastructures, MOS needs to be secured to ensure safety when communicating through or across untrusted networks.

As well, as we transition specific systems into the cloud or more IT-based networks, firewall restrictions are becoming more problematic. While VPN-based solutions may be suitable for some environments, for public cloud deployments and/or software as a service based or multi-tenant models, VPN is not a feasible solution, and proposes greater risk to the customer by opening their network to the outside world.

The goal of this proposal is to identify a means of adding a security-based implementation to the MOS protocol with minimal effort, and impact to the specification, or using proprietary technology outside of the control of this working group specification. Finding a way to properly traverse one-way firewalls is of secondary importance but makes sense to tackle at the same time given it has considerable implications for certain applications and deployments.

For an overview of web socket based technology, I recommend you review the following information to fully understand the scope of this proposal.

WebSocket Overview - https://en.wikipedia.org/wiki/WebSocket
About HTML5 WebSocket - https://www.websocket.org/aboutwebsocket.html
Jetty WebSocket Example - http://www.eclipse.org/jetty/documentation/9.4.x/jetty-websocket-server-api.html
Apache MosProxyWsTunnel - https://httpd.apache.org/docs/2.4/mod/mod_proxy_wstunnel.html
WebSocket Security - https://devcenter.heroku.com/articles/websocket-security

Proposal

I am proposing the adoption of SSL over a web socket based implementation for adding security to MOS. This is a standard, off the shelf, and well understood technology with a wide set of libraries for every language, and does not fundamentally change any of the other behavior of MOS. As well, with web-socket based implementations, we are proposing a behavior change to the protocol to support a new mode called “Passive Mode” as a requirement.

We are proposing this be called MOS v4.0 and can be used as a basis for future improvements to the specification.

As web sockets implicitly have message boundaries built in, the implementation is simpler than most typical MOS implementations. From a newsroom perspective, both the web socket and traditional socket models may be implemented, but for security perspectives, it should be noted that a user should be provided the option to disable the traditional socket (2.x) and web services (3.x) setup for security sensitive environments.

MOS Passive Mode is a proposal similar to FTP PASV, which allows for bi-directional MOS-based communication to originate from one end. In this proposal, we are outlining, via the use of the web socket based approach, a means of supporting this passive approach. In this case, one end (on the “inside” of the firewall) will establish both outbound and inbound connections to the external device/newsroom, allowing for inbound communication to happen but without having to open or expose firewall ports to do so.

Technical Details

In order for web socket-based communication to work, it requires the newsroom and device to expose an SSL/TLS-capable web server interface (generally on port 443, as a standard HTTPS, but this could be configured per device/newsroom as necessary). Each device and newsroom must also expose an HTTP/s “endpoint” supporting the web socket protocol, a URL in which the web socket communication occurs. This can be configured per device as well. For example, a newsroom “endpoint” might be:

```
wss://<NEWSROOMHOSTNAME>/mos/Communication
```

In order to start communicating via MOS to this newsroom endpoint, the client simply opens a Web Socket client to that endpoint. You can then issue web socket messages (MOS messages) to the device. Notice specifically the protocol being wss, this is web socket secure (similar to HTTPS vs HTTP). In non-secure environments, a "ws" schema can be used instead of wss for insecure web socket communication (thus no SSL certificates are required).

For WSS, similar to HTTPS, a certificate is required to be installed on each newsroom and device. In the case of MOS passive mode, where only one side is listening for communication, only a single SSL certificate for the "external" or "listening" device/newsroom is required. In fact, if passive mode is used, only the "listening" side needs to expose a web socket server at all.

In addition to the URL above, to ensure security at a "connection" level, parameters should be provided to the URL in order to authenticate the connection. At a minimum, a mosId and ncsID, and a channel (ro, mom, aux). In this case, the channel represents the old MOS port (mom=10540, ro=10541, aux=10542), in that the behavior of two ports four sockets doesn't fundamentally change. Clients are expected to use this URL to connect two sockets, one with channel=mom, and another with channel=ro.

As well, you should pass along any additional query parameters that are entered as part of the device/newsroom URL configuration (for example, a newsroom could request additional configuration if desired as a configuration parameter).

So, if a device was configured inside of a newsroom to use the URL

```
wss://<DEVICEHOSTNAME>/somepath/MyDeviceMos?parameter1=12345
```

Then the newsroom, when it needs to communicate to this, will open two web socket clients to:

```
wss://<DEVICEHOSTNAME>/somepath/MyDeviceMos?parameter1=12345&mosID=MYMOSID&ncsID=MYNCSID&channel=mom
```

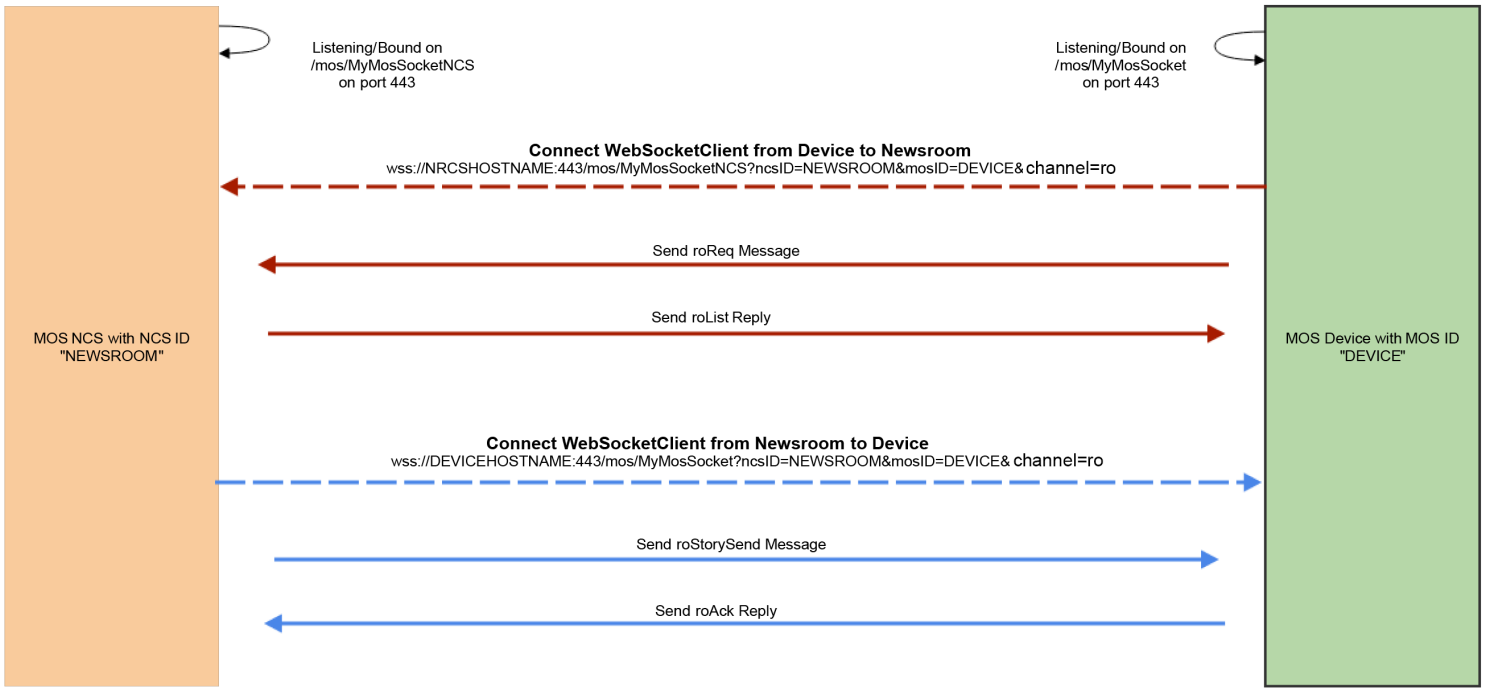
```
wss://<DEVICEHOSTNAME>/somepath/MyDeviceMos?parameter1=12345&mosID=MYMOSID&ncsID=MYNCSID&channel-ro
```

As with typical (now called “active”) MOS communication (without using MOS passive), both the device and the newsroom will open up two web socket clients each (one, which would have traditionally been port 10540, and one traditionally on port 10541, each) to the other side’s “web socket URL”, using the channel mom and channel ro respectively. You can then freely send messages using the web socket client/server model where each MOS message is automatically bounded by the web socket protocol.

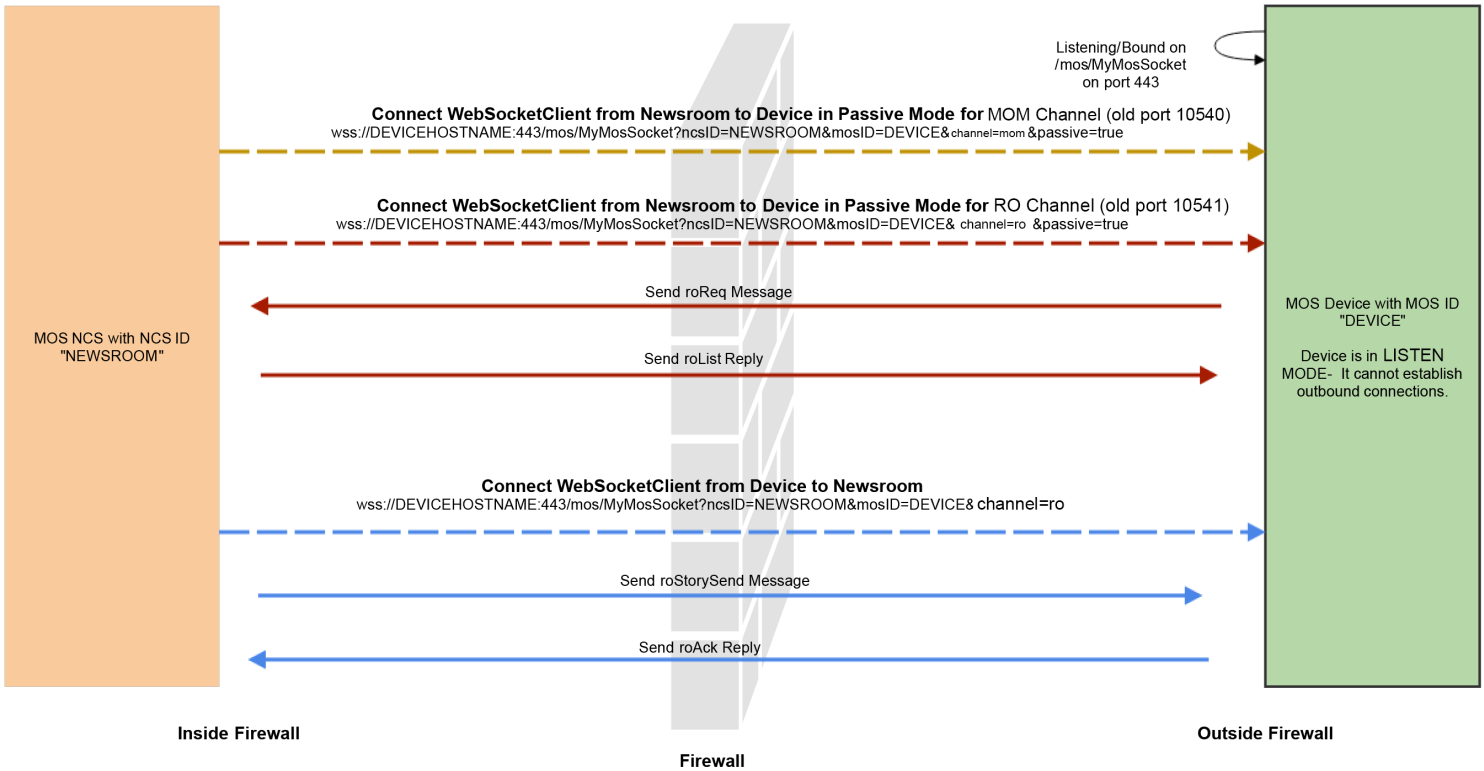
In the case of MOS passive, the device/newsroom on the “inside” of the firewall will open a web socket client and connect to the external device/newsroom using this wss URL. One additional parameter is added to the URL scheme, “passive=true”. This instructs the receiving device to treat it as an outbound socket. If the socket closes for any reason, the internal device must re-establish the socket as quickly as possible. One socket per port should be established. When the “external” device needs to originate a message sequence, for example an roReq message to the “internal” device, it will use this “passive” web socket connection for the specific port that it was provided.

A comparison of these workflows is described in the diagrams on the following page.

RoReq / RoList Sequence for Standard Web Socket Implementation



RoReq / RoList Sequence for Passive Web Socket Implementation



Note: Passive Connections Yellow and Red must remain connected at all times. If the connection disconnects, the" active" device(in this case, the newsroom) is expected to immediately reconnect them. Otherwise, inbound messages originating from the external device will not be received.

The beauty of this model is that all of the SSL and encryption are handled in a standard way, it is well understood and communicated to customers and security professionals, and there is a diverse set of libraries to work with.

> **Implementation Tip**
> *Implementing MOS via web sockets does not generally require re-writing your entire MOS stack. In our case, we simply added a WebSocket server to the applications using a standard library (in our case, Jetty Web Sockets). When we receive a web socket request, it can either be delegated and relayed internally to the same "queue" used when receiving standard MOS messages, or you could actually internally create a MOS "socket" bridge between the web socket and your typical socket and just relay messages. We've done both approaches depending on the product and both work successfully.*

Authentication

Due to the security and network conditions that MOS secure is designed to operate, all web socket channel communication should be secured. It is strongly recommended that devices use HTTPS when possible, and we are proposing the use of a username/password schema on the basis of the HTTP Authorization headers with the "Basic" schema. This is well defined and secure, and by not putting sensitive credentials into the URL avoids any accidental interception or capture of this data in logs, etc.

This requires an authorization fields to be added to the HTTP header on connection of the web socket.

From Wikipedia:

> *When the user agent wants to send authentication credentials to the server, it may use the Authorization field. The Authorization field is constructed as follows:*
>
> *1. The username and password are combined with a single colon. (:). This means that the username itself cannot contain a colon.*
>
> *2. The resulting string is encoded into an octet sequence. The character set to use for this encoding is by default unspecified, as long as it is compatible with US-ASCII, but the server may suggest use of UTF-8 by sending the charset parameter.*
>
> *3. The resulting string is encoded using a variant of Base64.*

> *4. The authorization method and a space (e.g. "Basic ") is then prepended to the encoded string. For example, if the browser uses Aladdin as the username and OpenSesame as the password, then the field's value is the base64-encoding of Aladdin:OpenSesame, or QWxhZGRpbjpPcGVuU2VzYW1l. Then the Authorization header will appear as:*
>
> *Authorization: Basic QWxhZGRpbjpPcGVuU2VzYW1l*

Required URL Parameters

In order for the web sockets to be secure and for communication to flow, the following parameters MUST be provided for each web socket connection request.

> ***mosID***: the MOS ID of the device connecting (or being connected to)
> ***ncsID***: the NCS ID of the newsroom connecting (or being connected to)
> ***channel***: the “channel” in which the MOS connection represents (this is indicative of the traditional port we would connect on, wherein the values “ro” for running order (10541), “mom” for media object metadata (10540), and “aux” for auxiliary (10542). Plus, any other parameters configured in the URL of the device or newsroom (such as API key, username, password, etc). These are secure as when communicated over HTTPS/SSL connections, the URLs are not transmitted in plain text, the security layer is established prior to communicating the HTTP request.

Each device/newsroom must provide a configurable endpoint exposing the entire URL, including the protocol (ws/wss), hostname, SSL port, path, and query string. While assumptions and defaults can be made, it must be available somewhere in the application to configure all of these parameters.

Other Notes

Two Ports / Four Sockets

This proposal does not change the way that MOS v2 socket management is handled. It remains a two port, four socket design, in that each port, inbound and outbound sockets are established. In the case of passive mode, the device in “active” mode established

two outbound sockets, one of which is to be used as an inbound socket and flagged accordingly.

Requirement for Passive Mode

It has been suggested that we make passive mode a requirement for devices in this updated design, in order to ensure that we can mitigate firewalls and other benefits of passive mode in the future. As such, it is a recommendation that the NRCS side must support the “active” side of passive mode (it establishes all connections to target devices, inbound and outbound) in order to be compliant with the specification. Optionally, the device can also support the “active” side of passive mode if desired.

This requirement has now been accepted and with web-based sockets one side will always need to be passive and one side active. It is clarified in the amendments in the appendix.

Self Signed and Untrusted Certificates

A discussion is to be had whether or not untrusted (or self signed) certificates may be valid in this implementation.

It is recommended that while authority issued certificates are always accepted, devices and newsrooms are expected to also accept untrusted (or self signed) certificates, or provide an option to do so. When configuring the device, if untrusted certificates aren’t enabled by default, a configuration option should be provided allowing for this option. As well, for security purposes, it is recommended that devices allowing untrusted by default provide an option to disable this.

The acceptance of untrusted or self signed certificates has been accepted with conditions as noted in the amendments in the appendix.

KeepAlive Message Addition to Profile 0

Through implementation and prototyping, one scenario was uncovered that has proven to be problematic to solve in the web socket implementation. That is, network firewalls and proxies have proven to disconnect web-sockets at a very short interval of non-traffic. While heartbeats can be configured bi-directionally at quick intervals, it adds a lot of overhead and timing concerns in doing so. In our experience, heartbeats can result

in significant protocol issues in devices or newsroom interfaces that have resulted in significant failures.

However, we need a keep alive mechanism on the web socket, especially when passive mode is used. As we have clear message boundaries which provide demarcation of each MOS message in a web socket based implementation, I am proposing a very simple solution for this. Each connection can (optionally) send keepalive messages, a new MOS message set in profile 0, which when received by the other end, are simply discarded, no reply necessary as is with a heartbeat.

As no reply is required, messageID is not necessary for this message as it is not sequenced.

```
<mos>
  <mosID>mosID</mosID>
  <ncsID>ncsID</ncsID>
  <keepAlive/>
</mos>
```

This message is valid on both MOS channels, bidirectionally. It should be discarded on receipt and no reply is necessary.

As per most implementations of a KeepAlive signal in Web Socket implementations, it is recommended this be sent at a duration of 30 seconds. However, it is also acceptable to only send this message when an idle period of 30 seconds on the socket has been detected to reduce traffic. This should be sent across each connected socket/channel at this interval to ensure that the connection is not severed.

MOS v2.0 End of Life

This specification is noted as a replacement for MOS v2.0. Whether it is called MOS v4.x is up for debate. It has been suggested that a period be defined where MOS v2.x is declared end of life once this specification is approved.

This specification will NOT dictate an end of life date for MOS v2.x.

Optional Request for Consideration

Removal of Deprecated Messages from MOS v4.0

A recommendation can be made to officially remove messages that were deprecated in the v2.x series of MOS

These messages would include the following:

roStoryAppend
roStoryInsert
roStoryReplace
roStoryMove
roStoryMoveMultiple
roStorySwap
roStoryDelete
roItemInsert
roItemReplace
roItemMoveMultiple
roItemDelete
roStat
roItemStat