MCOLLECTIVE 安全综述

作为 MCollective 重要部分就是考虑安全。默认情况下允许所有的代理和所有节点所有代理进行通信。问题是通过这种方法如果在某个节点有一个不受信任的用户他就能够安装一个客户端应用程序来从 server 配置文件中读取用户名/密码从而控制整个体系结构。本部分分析在 MCollective 使用过程中的安全问题，并提供几种安全方案。

通过 TLS 配置 MCOLLECTIVE 到 ACTIVEMQ 端到端加密

这种方式允许你使用 **MCollective** 的快速和有效的 **SSL** 安全插件而不是缓慢的难配置的 **AES** 安全插件。主要有两种方式来实现**:**

CA-verified TLS 对流量进行加密，也允许你限制中间件连接**-**仅仅那些来自站点 **CA** 并拥有有效证书的节点才能连接。

Anonymous TLS 消息进行加密，防止随意流量嗅探。，这会防止 **MCollective** 使用的密码被窃取。然而，它不会检查某节点是否被允许连接，这时只能相信用户名和密码的安全性。

下面介绍第一种方式 **TLS**，即 **CA-verified TLS,**这是一种更加推荐也更加安全的方式。配置过程分三步：

配置 ACTIVEMQ 使用 TLS

为 ActiveMQ 创建 keystores

Active 需要下列证书：

站点 CA 证书的副本

站点 CA 签名的证书

匹配证书的私钥

这些证书可以来自任何地方，但是必须匹配 MCollective 使用的 CA。最直接的方式是重用站点的 puppet 认证体系，因为它到处存在而且有工具来签名任意证书。无论如何，记得保护私钥。这里选择重用 puppet 认证。

获得证书和密钥

在 ActiveMQ Server 上：

```
sudo puppet agent --configprint ssldir
```

找到 SSL 目录后，将下列文件拷贝到工作目录，确保 cert key 和 private key 有不同的名字。

```
mkdir -p /opt/activemq/ssl

cd /opt/activemq/ssl/

cp /var/lib/puppet/ssl/certs/ca.pem .

cp /var/lib/puppet/ssl/certs/<name>.pem <name>_cert.pem

cp /var/lib/puppet/ssl/private_keys/<name>.pem <name>_private.pem
```

这里的<name>为 puppet certname

创建 truststore

Truststore 决定哪些证书被允许连接到 ActiveMQ。如果你导入某个 CA 认证，ActiveMQ 将会信任由这个 CA 签名的任何证书。命令中的 ca.pem 可以是任何 ca 认证的拷贝，在提示时设置 truststore 的密码。

```
keytool -import -alias "My CA" -file ca.pem -keystore truststore.jks
```

验证过程：

```
keytool -list -keystore truststore.jks

openssl x509 -in ca.pem -fingerprint -md5
```

创建 keystore

Keystore 包含 ActiveMQ 的证书和私钥，这用来向连接它的应用程序标识它自身。这些创建命令使用‘export/source’密码和‘destination’密码。export 密码在这些命令之后不会再用到。destination 密码是 keystore 密码。

```
cat <name>_private.pem <name>_cert.pem > temp.pem

openssl pkcs12 -export -in temp.pem -out activemq.p12 -name <name>

keytool -importkeystore -destkeystore keystore.jks -srckeystore activemq.p12 -
srcstoretype PKCS12 -alias <name>
```

验证过程：

```
keytool -list -keystore keystore.jks

openssl x509 -in puppet.example.com_cert.pem -fingerprint -md5
```

将 keystore 和 truststore 移至 ActiveMQ 配置目录，确保这些文件由 ActiveMQ 拥有并且对其他用户不可读。

配置 **activemq.xml** 的**<sslContext>**

```
    <sslContext>

      <sslContext
```

```
        keyStore="keystore.jks" keyStorePassword="secret"
        trustStore="truststore.jks" trustStorePassword="secret"
      />
</sslContext>
```

这个配置默认 keystore.jks 和 truststore.jks 文件与 activemq.xml 在同一目录，并且这里冗余的嵌套 sslContext 不是错误，而是语法所致。这里的密码是之前创建时的密码。

编辑完 activemq.xml 文件之后，需要仔细检查确保 activemq.xml world-unreadable,因为现在它包含敏感信息。

配置 transport connector 合适的 URI

CA-verified TLS 的默认 transportConnector 配置：

```
    <transportConnectors>
      <transportConnector name="stomp+ssl"
uri="stomp+ssl://0.0.0.0:61614?needClientAuth=true"/>
    </transportConnectors>
```

重启 **ActiveMQ**

```
/etc/init.d/activemq restart
/ect/init.d/activemq status #重启后观察，可能启动后又失败
```

此时，**MCollective Servers** 和 **Clients** 应该无法连接到 **ActiveMQ**，因为它们还没有配置证书。

配置 MCOLLECTIVE SERVERS

对于 **MCollective Deamon** 可以通过编辑 **server.cfg** 来使用已经有的 **puppet** 认证证书

```
connector = activemq
# Optional:
# plugin.activemq.base64 = yes
plugin.activemq.pool.size = 1
plugin.activemq.pool.1.host = stomp.example.com #Stomp Server Name
plugin.activemq.pool.1.port = 61614
plugin.activemq.pool.1.user = amq
plugin.activemq.pool.1.password = secret
```

```
plugin.activemq.pool.1.ssl = true

plugin.activemq.pool.1.ssl.ca = /var/lib/puppet/ssl/ca/ca_crt.pem

# 也可以是 /var/lib/puppet/ssl/certs/ca.pem

plugin.activemq.pool.1.ssl.key = /var/lib/puppet/ssl/private_keys/<name>.pem

plugin.activemq.pool.1.ssl.cert = /var/lib/puppet/ssl/certs/<name>.pem
```

注意这里的**<name>**是 **puppet certname** ，用户名和密码设置正确。

可以通过以下命令来获取这些证书和密钥。

```
puppet agent --configprint hostprivkey #匹配证书私钥

puppet agent --configprint hostcert #CA 签名证书

puppet agent --configprint localcacert #CA
```

重启服务验证生效。

```
/etc/init.d/mcollective restart

/etc/init.d/mcollective status #多查看几次，有可能启动后又失败
```

配置 MCOLLECTIVE CLIENTS

每个 Client 现在需要由 Puppet Ca 生成的证书从而能够连接到 ActiveMQ

可以采用 puppet cert generate 生成相应证书。如果已经有证书，可直接使用。

将证书拷贝到用户工作目录下：

```
mkdir -p /opt/mcollective/ssl

cd /opt/mcollective/ssl/

cp /var/lib/puppet/ssl/ca/ca_crt.pem .

cp /var/lib/puppet/ssl/private_keys/<name>.pem <name>_private.pem

cp /var/lib/puppet/ssl/public_keys/<name>.pem <name>.pem

cp /var/lib/puppet/ssl/certs/<name>.pem <name>_cert.pem
```

配置 client.cfg 来使用这些证书

```
connector = activemq

# Optional:

# plugin.activemq.base64 = yes
```

```
plugin.activemq.pool.size = 1

plugin.activemq.pool.1.host = master.example.com

plugin.activemq.pool.1.port = 61614

plugin.activemq.pool.1.user = amq

plugin.activemq.pool.1.password = secret

plugin.activemq.pool.1.ssl = true

plugin.activemq.pool.1.ssl.ca = /opt/mcollective/ssl/ca_crt.pem

plugin.activemq.pool.1.ssl.key = /opt/mcollective/ssl/<name>_private.pem

plugin.activemq.pool.1.ssl.cert = /opt/mcollective/ssl/<name>.pem
```

备注：如果使用 SSL 安全插件，可以通过设置/opt/mcollective/ssl/<name>.pem 作为公钥来使用相同的文件。

最后验证加密过程，采用 Tcpdump 抓包查看。

基于认证授权进行细粒度权限控制

默认的消息主题格式与 ActiveMQ Wildcard 模式相兼容，因此可以做细粒度的控制。

ACTIVEMQ WILDCARD 模式

ActiveMQ 支持目的地通配符来提供简单的支持联合的名称层次结构。这个概念在金融市场流行很长时间，用来作为一种以层次结构组织事件（如价格变化）的方式并且用也通配符能够很容易订阅你所感兴趣的信息范围。

假设想通过证券交易所订阅发布价格消息，可能会使用如下结构：

```
PRICE.STOCK.NASDAQ.ORCLE  发布 ORACLE 在 NASDAQ 的股票价格

PRICE.STOCK.NYSE.IBM    发布 IBM 在 NYK 证券交易所的股票价格
```

订阅者同样会使用相同目的地来订阅正需要的价格，或者使用通配符来定义层次模式匹配目的地。

通配符支持

使用如下通配符：

```
.用来作为路径分隔符

*用来匹配路径任意名称

>用来递归匹配任意以这个名称开始的目的地
```

例如：

```
订阅      含义

PRICE.> 任意交易所任意产品的任意价格

PRICE.STOCK.>    任意交易所一个股票的任意价格

PRICE.STOCK.NASDAQ.*     NASDAQ 的任意股票价格

PRICE.STOCK.*.IBM    任意交易所的任意 IBM 股票价格
```

ACTIVEMQ 中基于用户和组认证授权设置

当连接时，MCollective Clients 和 Servers 提供 username,password 和可选的 SSL 认证。ActiveMQ 可以使用它们来作验证。一般通过添加合适的元素至<plugins>元素中来设置验证。ActiveMQ 提供三种方式：simpleAuthenticationPlugin, jaasAuthenticationPlugin, jaasCertificateAuthenticationPlugin。simpleAuthenticationPlugin 在 activemq.xml 文件中直接定义用户，它也是最简单和广泛测试的一种方式，示例如下：

```
    <plugins>

      <simpleAuthenticationPlugin>

        <users>

          <authenticationUser username="mcollective" password="marionette"
groups="mcollective,everyone"/>

          <authenticationUser username="admin" password="secret"
groups="mcollective,admins,everyone"/>

        </users>

      </simpleAuthenticationPlugin>

      <!-- ... authorization goes below... -->

    </plugins>
```

这里创建了两个用户，期望 MCollective Server 以 mcollective 登陆，而 Admin 用户以 admin 登陆。注意到除非设置授权访问，否则这些用户拥有相同能力。

授权简单说就是给用户组设置权限。默认情况下，ActiveMQ 允许所有人对所有 topics 或 quene 拥有读权限和写权限，而 admin 能够创建 topic 或者 quene。

通过设置<authorizationPlugin>元素中的规则，可以调节权限控制。注意以下几点：

权限是通过组划分的，用户组是最小的权限划分单位。

MCollective 在它们知道将会接收到内容之前创建 subscription。也就是说任何能够对一个 destination 读写的角色一定对这个 destination 有 admin 权限。可以将 admin 作为读写的一个超集。

基本限制示例

```
    <plugins>

      <!-- ...authentication stuff... -->

      <authorizationPlugin>

        <map>

          <authorizationMap>

            <authorizationEntries>

              <authorizationEntry queue=">" write="admins" read="admins"
admin="admins" />

              <authorizationEntry topic=">" write="admins" read="admins"
admin="admins" />

              <authorizationEntry topic="mcollective.>" write="mcollective"
read="mcollective" admin="mcollective" />

              <authorizationEntry queue="mcollective.>" write="mcollective"
read="mcollective" admin="mcollective" />

              <authorizationEntry topic="ActiveMQ.Advisory.>" read="everyone"
write="everyone" admin="everyone"/>

            </authorizationEntries>

          </authorizationMap>

        </map>

      </authorizationPlugin>

    </plugins>
```

这意味 admin 能够发出命令而 MCollective server 能够读写和应答这些命令。但是它也意味着，MCollective server 能够发出命令，这可能不是你想看到的。

注意到组‘everyone’并不是特殊的，需要手动确保所有的用户都是这个组的一员。对于 everyone,ActiveMQ 没有类似 mcollective 这样的 wildcard。

详细限制示例

```
    <plugins>
```

```
      <!-- ...authentication stuff... -->

      <authorizationPlugin>

        <map>

          <authorizationMap>

            <authorizationEntries>

              <authorizationEntry queue=">" write="admins" read="admins"
admin="admins" />

              <authorizationEntry topic=">" write="admins" read="admins"
admin="admins" />

              <authorizationEntry queue="mcollective.>" write="admins"
read="admins" admin="admins" />

              <authorizationEntry topic="mcollective.>" write="admins"
read="admins" admin="admins" />

              <authorizationEntry queue="mcollective.nodes" read="mcollective"
admin="mcollective" />

              <authorizationEntry queue="mcollective.reply.>" write="mcollective"
admin="mcollective" />

              <authorizationEntry topic="mcollective.*.agent" read="mcollective"
admin="mcollective" />

              <authorizationEntry topic="mcollective.registration.agent"
write="mcollective" read="mcollective" admin="mcollective" />

              <authorizationEntry topic="ActiveMQ.Advisory.>" read="everyone"
write="everyone" admin="everyone"/>

            </authorizationEntries>

          </authorizationMap>

        </map>

      </authorizationPlugin>

    </plugins>
```

这样的设置，admin 能够发出命令而 MCollective server 能够读写和应答这些命令。但是 server 不能发出命令，这取决于 mcollective.registration.agent destination。

对于 admin，能够读写命令，因为它们对所有的出口 mcollective. destination 设置。admin 控制着整个体系。

ActiveMQ 与 MCollective server/client 的配置时对应的。所以最后对于节点也需要相应配置，在

server.cfg 中至少有以下几项：

```
plugin.stomp.user = mcollective

plugin.stomp.password = pI1SkjRi

plugin.psk = aBieveenshedeineeceezaeheer
```

对于客户端，建议用户详细信息配置使用 shell 环境变量：

```
export STOMP_USER= mcollective

export STOMP_PASSWORD= pI1SkjRi

export STOMP_SERVER=stomp1

export MCOLLECTIVE_PSK=aBieveenshedeineeceezaeheer
```

最后当用户 mcollective 登录到一个拥有这些环境变量的 shell 时会获得它所属用户组对于不同命令的相应权限。

SUBCOLLECTIVE 的安全设计

Subcollective 在 MCollective 中是个非常有用的设计。除了网络分区的作用之外，在安全方面也是有很多考虑。虽然在 SimpleRPC 框架中有一个安全模型能够注意拓扑结构但核心网络层没有。即使你仅仅给某人权限来执行 SimpleRPC 请求，他依然可以使用 mco ping 来发现网络中其他节点。

通过创建 submcollectives 你可以有效而且完备地在现有网络上创建隔离区，并在中间件层给予限制。这些限制必须在中间件 server 上进行配置，而不是 MCollective 自身，所以方式会随着使用的中间件不同而变化。下面是 ActiveMQ 的推荐配置。

结合 SUBCOLLECTIVES 的细粒度权限控制

之前例子只是针对单一的 collective：mcollective。这在 mcollective 中是默认的 main_collective。如果你设置了其他的 subcollectives。它们的目的地将会以它们的名字开始而不是 mcollective。如果你需要对每个 collective 单独控制。可以使用类似以下的模板设置。

```
    <plugins>

      <!-- ...authentication stuff... -->

      <authorizationPlugin>

        <map>

          <authorizationMap>

            <authorizationEntries>

              <!-- "admins" group can do anything. -->

              <authorizationEntry queue=">" write="admins" read="admins"
admin="admins" />
```

```
            <authorizationEntry topic=">" write="admins" read="admins"
admin="admins" />



            <%- @collectives.each do |collective| -%>

            <authorizationEntry queue="<%= collective %>.>" write="admins,<%=
collective %>-admins" read="admins,<%= collective %>-admins" admin="admins,<%=
collective %>-admins" />

            <authorizationEntry topic="<%= collective %>.>" write="admins,<%=
collective %>-admins" read="admins,<%= collective %>-admins" admin="admins,<%=
collective %>-admins" />

            <authorizationEntry queue="<%= collective %>.nodes"
read="servers,<%= collective %>-servers" admin="servers,<%= collective %>-servers"
/>

            <authorizationEntry queue="<%= collective %>.reply.>"
write="servers,<%= collective %>-servers" admin="servers,<%= collective %>-
servers" />

            <authorizationEntry topic="<%= collective %>.*.agent"
read="servers,<%= collective %>-servers" admin="servers,<%= collective %>-servers"
/>

            <authorizationEntry topic="<%= collective %>.registration.agent"
write="servers,<%= collective %>-servers" read="servers,<%= collective %>-servers"
admin="servers,<%= collective %>-servers" />

            <%- end -%>

            <authorizationEntry topic="ActiveMQ.Advisory.>" read="everyone"
write="everyone" admin="everyone"/>

          </authorizationEntries>

        </authorizationMap>

      </map>

    </authorizationPlugin>

  </plugins>
```

这个例子中将用户分为以下用户组：

Admins 是‘super-admins’组，能控制所有的 servers

Servers 是‘super-server’组，能够接收和响应所有的它们认为是其中一员的 collective 的命令

COLLECTIVE-admins 仅仅能够命令它们特定的 collective。（因为所有的 servers 可能都会是默认的 collective: mcollective，那么 mcollective-admins 组某种意义上类似‘super-admins’）

COLLECTIVE-servers 能够仅仅接收和响应它们特定的 collective

注意的是，这些 subcollective 同时需要在 servers 和 clients 端进行相应的配置。

针对 ACTIVEMQ 集群的目的地流量过滤

假想一个星型的网络拓扑。有 uk_collective,us_collective,zh_collective 三个集群（它们各有一个中心 broker）连接至同一个中心 broker。如果希望 center_broker 至 uk_broker 的连接仅仅传输 mcollective collective 和 uk 特定的 uk_collective collective 的流量。可以设置如下：

```
    <dynamicallyIncludedDestinations>
      <topic physicalName="mcollective.>" />
      <queue physicalName="mcollective.>" />
      <topic physicalName="uk_collective.>" />
      <queue physicalName="uk_collective.>" />
    </dynamicallyIncludedDestinations>
```

在这个配置中，连接至中心 broker 的 admin 用户能够对 uk_collective 中的节点发布命令，但是连接至 uk broker 的 admin 用户不能对 us_collective 中的节点发布命令。

换过来，如果 uk broker 连接至中心 broker，但希望其他不是 uk_collective 的节点忽略 uk_collective 特定的流量，可以配置忽略 uk_collective 目的地：

```
    <excludedDestinations>
       <topic physicalName="uk_collective.>" />
       <queue physicalName="uk_collective.>" />
    </excludedDestinations>
```

在这种情况下，连接至中心 broker 的 admin 用户无法向 uk_collective 中的节点发布命令。也就谁说它将通过 mcollective collective 来发布命令来跨过它们的界限。