# Apache ActiveMQ LDAP Wildcard Interpretation

## 06/03/2015

| | |
|---|---|
| **Software:** | ActiveMQ |
| **Affected Versions:** | ActiveMQ 5.0.0 - 5.10.0 |
| **CVE Reference:** | CVE-2014-3612 |
| **Author:** | Georgi Geshev - MWR Labs (http://labs.mwrinfosecurity.com/) |
| **Severity:** | High |
| **Vendor:** | Apache |
| **Vendor Response:** | Fix Released |

## Description:

Apache ActiveMQ is an open source message-oriented middleware messaging broker. ActiveMQ supports various message queueing protocols, such as Message Queue Telemetry Transport (MQTT), Advanced Message Queuing Protocol (AMQP), and Streaming Text Oriented Messaging Protocol (STOMP).

Two vulnerabilities were identified in the way Apache ActiveMQ handles client credentials when performing authentication with an LDAP server.

## Impact:

The vulnerabilities allow for leveraging the unauthenticated authentication mechanism, when supported by the remote LDAP service, or abuse an LDAP wildcard expansion weakness. The unauthenticated authentication mechanism, as specified in RFC 4513, may be used for performing unauthenticated Bind with an LDAP service. The wildcard interpretation weakness allows for brute forcing a password, for an unknown but valid account, as opposed to brute forcing a combination of username and password.

## Cause:

ActiveMQ was found to lack input sanitisation of the submitted client credentials before using them to authenticate with an external LDAP server.

## Interim Workaround:

The unauthenticated authentication mechanism vulnerability can be mitigated by disabling the respective mechanism on the LDAP server.

## Solution:

Upgrade to Apache ActiveMQ version 5.11.0 or later.

## Technical details:

Apache ActiveMQ provides pluggable security through various different providers. One of these providers is the ActiveMQ Java Authentication and Authorization Service (JAAS).

The LDAP authentication module, as implemented in ActiveMQ JAAS, was found to allow the unauthenticated authentication mechanism as specified in RFC 4513, when the latter is supported and enabled on the LDAP server. This mechanism allows for establishing an anonymous authorisation state by sending an LDAP Bind request with a valid non-zero length username and an empty password.

The security fix applied by Apache includes the following checks for non-null usernames and passwords before attempting to authenticate with the LDAP Server.

```
diff -u -N -re1bbde7302d7d169bec978633d2dd85eba2e19cc -
r0b5231ada5ce365b41832ba8752ee210145d1cbe
--- activemq-
broker/src/main/java/org/apache/activemq/network/LdapNetworkConnector.java
        (.../LdapNetworkConnector.java)        (revision
e1bbde7302d7d169bec978633d2dd85eba2e19cc)
+++ activemq-
broker/src/main/java/org/apache/activemq/network/LdapNetworkConnector.java
        (.../LdapNetworkConnector.java)        (revision
0b5231ada5ce365b41832ba8752ee210145d1cbe)
@@ -210,8 +210,16 @@
              env.put(Context.SECURITY_AUTHENTICATION, "none");
          } else {
              LOG.debug("    login credentials [{}:******]", user);
-             env.put(Context.SECURITY_PRINCIPAL, user);
-             env.put(Context.SECURITY_CREDENTIALS, password);
+             if (user != null && !"".equals(user)) {
+                 env.put(Context.SECURITY_PRINCIPAL, user);
+             } else {
+                 throw new Exception("Empty username is not allowed");
+             }
+             if (password != null && !"".equals(password)) {
+                 env.put(Context.SECURITY_CREDENTIALS, password);
+             } else {
+                 throw new Exception("Empty password is not allowed");
```

```
+             }
          }
          boolean isConnected = false;
          while (!isConnected) {
```

Another vulnerability was related to lack of input sanitisation when processing usernames supplied by MQ clients.

When LDAP authentication is enabled, it is possible for an attacker to supply an LDAP wildcard character instead of a valid username. The wildcard, when processed by the LDAP server, will be matched against a valid username. This allows an adversary to brute force a password for an unknown but valid account as opposed to brute forcing a combination of username and password. Once a valid password is found, the attacker can successfully authenticate with LDAP and publish/subscribe to a queue.

The vulnerability lies in the ActiveMQ implementation of the 'Find and Bind' LDAP authentication mode.

ActiveMQ takes the username supplied by a client and uses it to construct a search filter. LDAP search is then performed starting at a pre-configured base Distinguished Name (DN) and looking for the user DN. When a match is found, the returned DN is subsequently used for an LDAP Bind with a client provided password. When the initial search is performed using an LDAP wildcard, a list of all currently configured users is returned in response. In this case, ActiveMQ will process the first element of the list, extract its DN and proceed with a Bind attempt.

The fix applied by Apache escapes characters with special meaning before performing the initial user DN search.

```
diff -u -N -r20747eedca6577af2031c107d4b1ec3b69724731 -
r0b5231ada5ce365b41832ba8752ee210145d1cbe
--- activemq-
jaas/src/main/java/org/apache/activemq/jaas/LDAPLoginModule.java
        (.../LDAPLoginModule.java)     (revision
20747eedca6577af2031c107d4b1ec3b69724731)
+++ activemq-
jaas/src/main/java/org/apache/activemq/jaas/LDAPLoginModule.java
        (.../LDAPLoginModule.java)     (revision
0b5231ada5ce365b41832ba8752ee210145d1cbe)
@@ -190,7 +190,7 @@
          try {

              String filter = userSearchMatchingFormat.format(new String[] {
-                 username
+                 doRFC2254Encoding(username)
              });
              SearchControls constraints = new SearchControls();
              if (userSearchSubtreeBool) {
@@ -319,7 +319,7 @@
              return list;
          }
          String filter = roleSearchMatchingFormat.format(new String[] {
-             doRFC2254Encoding(dn), username
```

```
+                doRFC2254Encoding(dn), doRFC2254Encoding(username)
          });

          SearchControls constraints = new SearchControls();
@@ -459,9 +459,14 @@
              env.put(Context.INITIAL_CONTEXT_FACTORY,
getLDAPPropertyValue(INITIAL_CONTEXT_FACTORY));
              if (isLoginPropertySet(CONNECTION_USERNAME)) {
                  env.put(Context.SECURITY_PRINCIPAL,
getLDAPPropertyValue(CONNECTION_USERNAME));
+             } else {
+                 throw new NamingException("Empty username is not allowed");
              }
+
              if (isLoginPropertySet(CONNECTION_PASSWORD)) {
                  env.put(Context.SECURITY_CREDENTIALS,
getLDAPPropertyValue(CONNECTION_PASSWORD));
+             } else {
+                 throw new NamingException("Empty password is not allowed");
              }
              env.put(Context.SECURITY_PROTOCOL,
getLDAPPropertyValue(CONNECTION_PROTOCOL));
              env.put(Context.PROVIDER_URL, getLDAPPropertyValue(CONNECTION_URL));
@@ -484,7 +489,7 @@

     private boolean isLoginPropertySet(String propertyName) {
        for (int i=0; i < config.length; i++ ) {
-               if (config[i].getPropertyName() == propertyName &&
config[i].getPropertyValue() != null)
+               if (config[i].getPropertyName() == propertyName &&
(config[i].getPropertyValue() != null &&
!"".equals(config[i].getPropertyValue())))
                                    return true;
        }
        return false;
```

## Detailed Timeline

| Date: | Summary: |
|---|---|
| 13/08/2014 | Reported to Apache |
| 13/08/2014 | Apache confirms reception |

| | |
|---|---|
| 26/08/2014 | Apache requests a proof of concept |
| 26/08/2014 | MWR provides a proof of concept |
| 08/09/2014 | Apache suggests fix |
| 05/02/2015 | Public fix released |
| 06/03/2015 | Advisory published |