JBoss Arbitrary code execution via unrestricted deserialization in ReadOnlyAccessFilter (CVE-2017-12149)

Gal Goldshtein, 2017-31-12

In late August 2017 Redhat have published a security advisory regarding an arbitrary code execution vulnerability in JBoss and recently a Proof of Concept exploit was publicly released. This vulnerability is added to the long list of unsafe deserialization vulnerabilities discovered this year.

The vulnerable code is part of the HTTP Invoker service that provides HTTP and Remote Method Invocation (RMI) access. This service was first introduced in JBoss Application Server version 3.0.3 (which was released in September 2002) and is installed by default on instances based on versions prior to 7.0.0.

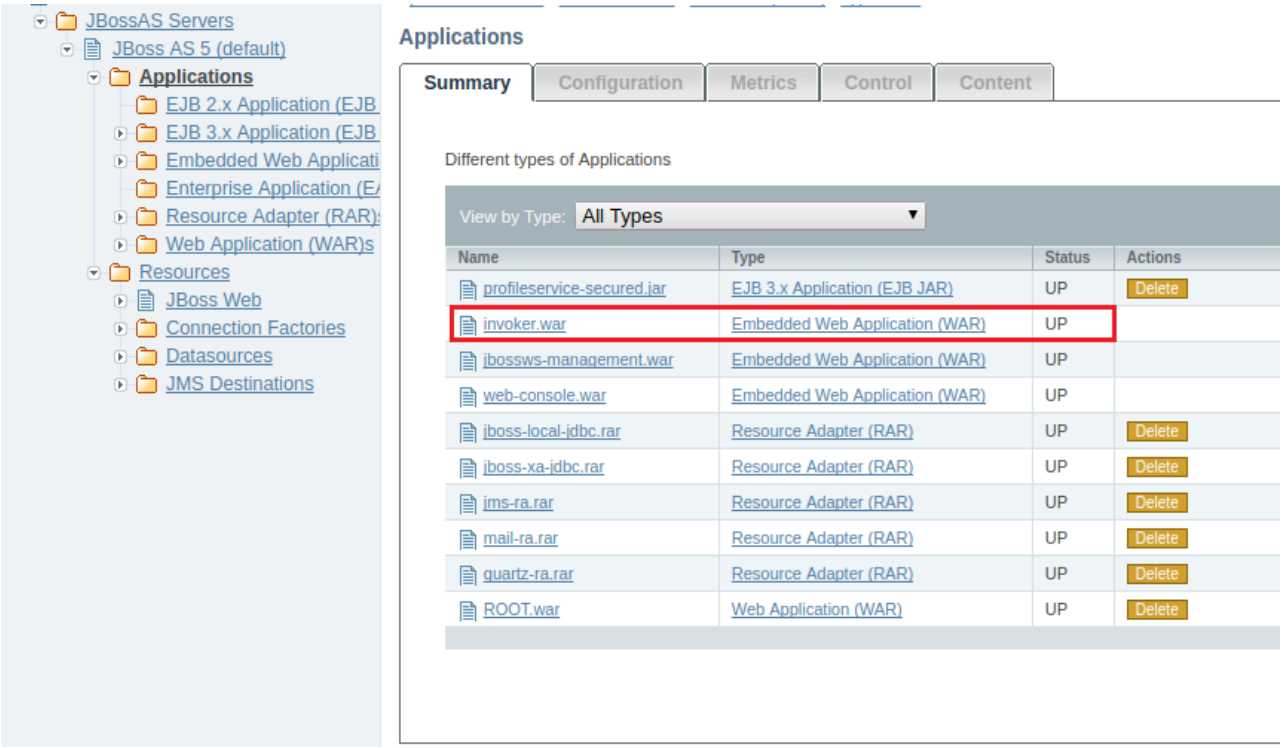

Figure 1: *invoker.war package pre-installed on JBoss Application Server 5.*

The unsafe deserialization takes place in the ReadOnlyAccessFilter.java file which receives a request object and calls readObject on the POST data sent by the user without doing any validations on the user supplied input. This provides attackers the possibility to send a crafted serialized object to the server that once deserialized will trigger arbitrary code execution in the context of the user running the vulnerable JBoss server.

```
public void doFilter(ServletRequest request, ServletResponse response,
   FilterChain chain)
   throws IOException, ServletException
{
   HttpServletRequest httpRequest = (HttpServletRequest) request;
   Principal user = httpRequest.getUserPrincipal();
   // If there was a read-only context specified validate access
   if( user == null && readOnlyContext != null )
   {
      // Extract the invocation
      ServletInputStream sis = request.getInputStream();
      ObjectInputStream ois = new ObjectInputStream(sis);
      MarshalledInvocation mi = null;
      try
      {
         mi = (MarshalledInvocation) ois.readObject();
      }
      catch(ClassNotFoundException e)
      {
         throw new ServletException("Failed to read MarshalledInvocation", e);
      }
      request.setAttribute("MarshalledInvocation", mi);
      /* Get the invocation method. If there is no method then this must
         be an invocation on an mbean other than our invoker so let it go
      */
      mi.setMethodMap(namingMethodMap);
      Method m = mi.getMethod();
      if( m != null )
         validateAccess(m, mi);
   }

   chain.doFilter(request, response);
}
```

Figure 2: *User supplied input is being deserialized without any validations being made on it.*

```
POST /invoker/readonly HTTP/1.1
Host: 127.0.0.1:8080
Connection: close
Accept-Encoding: gzip, deflate
Accept: */*
User-Agent: python-requests/2.11.1
Content-Length: 1465

[illegible] sr java.util.HashSet[illegible]4 xpw ?@ sr 4org.apache.commons.collections.keyvalue.TiedMapEntry[illegible]xs[illegible] L keyt Ljava/lang/Object;L mapt Ljava/util/Map;xpt foosr *org.apache.commons.collections.map.LazyMapnl
 L factoryt ,Lorg/apache/commons/collections/Transformer;xpsr :org.apache.commons.collections.functors.ChainedTransformer0[illegible](z[illegible] [
```

Figure 3: *Part of the POST request sent by the Proof-of-Concept exploit.*

Mitigation Using BIG-IP ASM

BIG-IP ASM customers under any supported BIG-IP version are already protected against this vulnerability. The exploitation attempt will be detected by existing Java code injection and command execution attack signatures which can be found in signature sets that include “Command Execution” and “Server Side Code Injection” attack types or “Java Servlets/JSP” System.

Detected Keyword	rs.ConstantTransformerXv[illegible]0x11A0x2[illegible]0x20x20x0]L0x00x9iConstantq0x0~0x00x3xpvr0x00x11java.lang.Runtime0x00x00x00x00x00x00x00x00x00x00x0xpsr0x0:org.apache.commons.collections.functor
Attack Signature	**Signature ID** 200003437 **Signature Name** Java code injection - java/lang/Runtime (Parameter)
Context	AMF Body
Applied Blocking Settings	Block Alarm Learn

Figure 4: *Exploit blocked with Attack Signature (200003437)*

Detected Keyword	0x0 0x0 0x0 0x0 0x0 xpvq 0x0 ~ 0x0 0x18 sq 0x0 ~ 0x0 0x1 3 ur 0x0 0x13 [Ljava.lang.String; V 0x1d {G 0x2 0 x0 0x0 xp 0x0 0x0 0x0 0x1 t 0x0 5/ bin/bash 0x20 -c 0x20 to 0x0 0x4 ex ecuq 0x0 ~ 0x0 0x1b
Attack Signature	**Signature ID** 200003057 **Signature Name** "/bin" execution attempt (Parameter)
Context	AMF Body
Applied Blocking Settings	Block Alarm Learn

Figure 5: *Exploit blocked with Attack Signature (200003057)*

Detected Keyword	mmons/collections/Transformer;xpsr 0x0 org.apach e.commons.collections.functors.ChainedTransform er 0U (z 0x4 0x2 0x0 0x1 I 0x0 0xa iTransformerst o x0 -[Lor
Attack Signature	**Signature ID** 200004297 **Signature Name** Java code injection - org/apache/commons/colle ctions/functors/ChainedTransformer
Context	AMF Body
Applied Blocking Settings	Block Alarm Learn

Figure 6: *Exploit blocked with Attack Signature (200004297)*

Detected Keyword	0x0 8 0x0 * org.apache.commons.collections.map.LazyM ap n 合 y 0x10 0x3 0x0 0x1 L 0x0 0x7 factoryt 0x0 .L org/apache/common
Attack Signature	**Signature ID** 200004298 **Signature Name** Java code injection - org/apache/commons/colle ctions/map/LazyMap
Context	AMF Body
Applied Blocking Settings	Block Alarm Learn

Figure 7: *Exploit blocked with Attack Signature (200004298)*

Detected Keyword	0x0 0x11 java.lang.Runtime 0x0 0x0 0x0 0x0 0x0 0x0 0x0 0x0 0x0 0x0 0x0 0x0 0x0 xpsr 0x0 : org.apache.commons.collections.functors.InvokerTransformer ���k[8 0x2 0x0 0x3] 0x0 0x5 iArgst 0x0 0x13 [Ljava/lang/
Attack Signature	**Signature ID** 200004299 **Signature Name** Java code injection - org.apache.commons.collections(4).functors.InvokerTransformer (Parameter)
Context	AMF Body
Applied Blocking Settings	Block Alarm Learn

Figure 8: *Exploit blocked with Attack Signature (200004299)*

F5 Networks, Inc. | 401 Elliot Avenue West, Seattle, WA 98119 | 888-882-4447 | f5.com

F5 Networks, Inc.
Corporate Headquarters
info@f5.com

F5 Networks
Asia-Pacific
apacinfo@f5.com

F5 Networks Ltd.
Europe/Middle-East/Africa
emeainfo@f5.com

F5 Networks
Japan K.K.
f5j-info@f5.com