CVE-2020-9548
Essay for course “Computer Security”

Author:
Edgar Selihov
Supervisor: Meelis Roos

Tartu 2020
Table of Contents

INTRODUCTION ........................................................................................................................................3
REPRODUCE JACKSON-BASED “GADGET” EXPLOIT ........................................................................3
POLYMORPHIC TYPING MEANING ....................................................................................................4
HOW TO PROTECT YOUR SYSTEM ..................................................................................................6
USED SOURCES ..................................................................................................................................7
Introduction

Back in 2017, many vulnerabilities regarding “serialization gadgets” were reported, which covered a wide range of Java serialization frameworks and libraries including today’s guest - Jackson.

Basically, “Gadgets” are classes which let execute inaccessible methods or gain access to non-accessible data when they are created or modified with the help of setters and/or assignments to fields. It is needed when reading data into Object (deserialization) or while writing of Objects into data (serialization).

If everything was initiated properly, these “Gadgets” are able to perform operations as a side effect, which means that remote scripts or commands may be executed and/or get information from the file system of server handling the deserialization.

Let’s imagine such a scenario. We have an imaginary method in our program named `setTimeToDetonate(int seconds)` which is an instance of a class Bomb. And when this method is called our JVM or even Operating System shuts down after specific delay. If our serialization system allows to construct and call `setTimeToDetonate`, then it can be potentially dangerous as this would be a valid denial-of-service attack (DoS) as most gadgets require calling of setters in specific order and then allow either to read of a system file or execute a code.

Reproduce Jackson-based “Gadget” exploit

1) Accept JSON content sent from an untrusted source, which was not written by you or was composed manually by attacker and you cannot restrict JSON itself that is being sent;
2) You have at least one specific “Gadget” set in your Java classpath and any class which works with Jackson;
3) Enable polymorphic type handling for properties with nominal type of `java.lang.Object` (or one of small number of “permissive” tag interfaces; `java.util.Serializable, java.util.Comparable`)
4) Use version of Jackson which does not block “Gadget” class yet.

NB!
(1) If you control both endpoints, this exploit won’t be available. But let’s pretend that this is a starting point.
Polymorphic Typing Meaning

Let’s consider an example, what “polymorphic handling” means. By default, certain types used must be declared by POJO (Plain Old Java Object) or the caller, so you use classes as in this example:

```java
public class Driver {
    public String name;
    public int age;
    public String nationality;
    public Transport transport; // embedded POJO
}

public class Transport {
    public enum fuel;
    public int totalSeatsNumber;
    // We are not going to include all the getters an
    setters..
}

Usually, deserializer knows well how to handle values for matching JSON values. Here is the example of an incoming JSON data:

```
{
    "name": "Edgar",
    "age": "25",
    "nationality": "Estonian",
    "transport": {
        "fuel": "petrol",
        "totalSeatsNumber": 4
    }
}
```

But as practices show, in OOP polymorphism is widely used for those purposes. In above example we could not define exact transport type, because it could be a car, or a bus, or a bike etc. Abstract classes solve this issue:

```java
abstract class Transport {
    public enum fuel;
    public int totalSeatsNumber;
}

public class Car extends Transport {
    public BigDecimal weight;
}
```
public class Bus extends Transport {}

If we want to serialize the data, it won’t throw any exceptions, because we have already data in classes stored. But if we try to deserialize incoming JSON data, it will cause exception, as Driver is abstract type and it cannot create instances and we must use one of subtypes. Deserializer just do not know what of subtypes it needs to use to properly deserialize data. Jackson can support this if we allow or require addition of “type id”: a value that allows deserializer to know which subtype to create instance of. There are multiple choices of adding “type id” to the subtype, but let’s consider only @JsonTypeInfo.

public class Driver {
    // Enable polymorphic handling only for this property
    @JsonTypeInfo(use = id.CLASS)
    public Transport transport;
}

By doing this, serializer will add a type identifier in JSON and deserializer knows what to expect in this type:

// How it will look like:

{
    "transport": {
        "@class": "package.Car",
        "fuel": "petrol",
        "totalSeatsNumber": 4
    }
}

Very important key part here is that only if class name is used as the type id, then a potential attacker can insert any class name he/she wants to: within limits of type compatibility Jackson will try to create an instance of that type and initialize it by calling setter methods and/or setting values to fields.
How to protect your system

1) Keep Jackson version up to date;
2) Avoid enabling default typing;
3) Avoid using java.lang.Object or java.util.Serializable as the nominal type of polymorphic values;
4) Use “type name” instead of “type id”.

Used Sources

1) https://nvd.nist.gov/vuln/detail/CVE-2020-9548
2) https://github.com/FasterXML/jackson-databind/issues/2634;
3) https://medium.com/@cowtowncoder/on-jackson-cves-dont-panic-here-is-what-you-need-to-know-54cd0d6e8062;
4) https://github.com/FasterXML/jackson-docs/wiki/JacksonPolymorphicDeserialization