Insecure Deserialization explained with examples

July 5, 2020 by thehackerish

Hello ethical hackers and welcome to this new episode of the OWASP Top 10 vulnerabilities series. In this blog post, you will learn Insecure Deserialization vulnerability. The plan is as follows:

- Insecure deserialization definition: This where you will learn the key terminologies and concepts behind this vulnerability,
- Examples of insecure deserialization in different programming languages: We will explore simple examples on PHP, Python and JAVA to help you understand how the vulnerability works.
- What is the impact: In this section, you will understand how bad insecure deserialization can be.
- Are there any real-world examples? In this section, we will explore many known CVEs which exploited this vulnerability. Some of them are insecure deserialization bug bounty reports from Hackerone.
- How to exploit it? This is where you will learn to go beyond running tools. You will perform white box testing and build custom code to solve a challenge on OWASP WebGoat.
- Finally, we will talk about how to mitigate insecure deserialization.

What is insecure deserialization?

Let's first understand the whole picture here. When you learn a programming language, the first thing you learn is how to define variables, classes and data structures that best suit your needs. Then, you learn how to manipulate them to achieve your needs. So far, they reside in memory, but sometimes, you need to store their states or share them with other systems. That's where serialization and deserialization come into play.

What is Serialization?

Let's say that you are playing with a character in a game. While you see the character on the screen, the software sees and manipulates an object residing in memory.

What if the game wants to store the state of that character in a file or share it with other systems? There should be a way to transform the in-memory object into a stream of bytes which can be easily stored and shared. That is what the process of serialization is all about. When the game performs the serialization of an object, we say that the object is serialized.

What is Deserialization?

Deserialization is the opposite of serialization. In fact, it consists of converting the serialized data into an in-memory representation which the software can then manipulate. Continuing on the previous example, when the game wants to retrieve the state of the serialized character object, it needs to deserialize it first.

What can go wrong here?

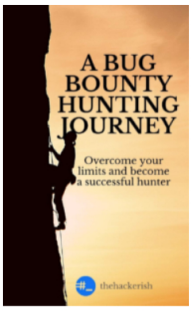
When a software deserializes user-controlled data without verification, we call it insecure deserialization. In our game example, an attacker might store a serialized file representing a malicious payload. If the developer doesn't perform a verification before deserialization, the insecure deserialization will trigger the attacker's code.

Insecure deserialization examples

Get you Free hacking lab VM

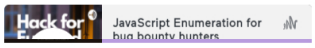
Click on the image and get all you need to kickstart your journey in Web Hacking!

Overcome your struggles and become a successful bug bounty hunter!



Click on the image and grab your own copy NOW!

Listen to the Hack for Fun and Profit Podcast



Support my work!

All the code discussed in this section is hosted on GitHub. Feel free to download it and follow along.

You find insecure deserialization in so many programming languages and technologies. We will explore it in PHP, Python and Java. I highly recommend you follow along with these three exercises on your own machine. If you don't have one, feel free to download the one I prepared for this course.

Insecure Deserialization in PHP

Also known as PHP object injection, this vulnerability takes advantage of PHP magic functions, like __destruct. Simply put, the vulnerable code defines some dangerous code of a certain class in this function. Then, it performs insecure deserialization somewhere. For example, the file-php-object-injection-example-deserialize-php file contains PHP code which deserializes an arbitrary object from a file named "serial". Notice that we have two classes: NormalClass which the developer expects, and DangerousClass which resides in the code.

However, an attacker sees this PHP insecure deserialization operation and crafts this code, which will serialize an object of the DangerousClass type, which runs the command "ls" by the application.

Let's spin up a new Docker container from our lab VM and serialize the malicious payload. We will map the `/tmp` directories of both the host and the guest. That way, we save the malicious file for later.

```
docker run -v /tmp:/tmp:rw -it php
```

Then, copy paste the gist which will perform the serialization and save the malicious file in the host's `/tmp` directory. From there, we will spin up a new PHP container which will simulate our vulnerable machine.

```
docker run -v /tmp:/tmp:rw -it php
```

Then, copy paste the gist. Notice that the "ls" command has run!

If we try to unserialize the variable `$serial` without first declaring the class in the vulnerable machine, we wouldn't be able to list the current directory. To exploit PHP deserialization using the unserialize function, there are two requirements:

- The vulnerable server has to define a class which define a `__destruct` function that runs dangerous code
- The attacker should be able to control the unserialized data.

Another form of PHP object injection is Phar deserialization. Basically, PHP Archives store the file metadata as serialized objects. When PHP code tries to perform certain operations on a file, the attacker's code will run. This Hackerone report is an example of that.

Insecure Deserialization in Python

In python, the insecure deserialization pickle vulnerability is overwhelmingly simple! It is sufficient to locate a feature which uses your input in `pickle.loads(user_input)`. Then, craft a code which will generate a serialized payload defining the `__reduce__` method. Let's look at this example:

```
import pickle

with open('serial', 'r') as f:
    pickle.loads(f.read())
```

This simple code performs a Python insecure deserialization from a file named `serial` using the Pickle module. If you go to the module documentation, you can see a red notice which clearly warns you not to deserialize untrusted data. The attacker sees my vulnerable code and generates a serialized payload which will run the OS command "id".

Now, when you run the vulnerable code, you will get the result of the command "id".

Java deserialization vulnerability example

In this insecure deserialization java example, we will explore, step-by-step, how we can exploit insecure deserialization in Java. The code defines a class with the name `NormalObj`, which does nothing but print the `name` attribute when the deserialization happens. This behaviour is defined in the `readObject` method. However, there is also another class that I called `VulnObj`, which is not called anywhere in the code. The `VulnObj` class defines a `readObject` method which runs arbitrary commands when the deserialization happens.

Unfortunately, I assumed that nobody will access the filesystem. Therefore, I blindly performed deserialization from the file `normalObj.serial` without verification.

An attacker sees this buggy code and crafts this program. It serializes a malicious `VulnObj` object containing the command `ls`. Then, it stores it on disk with the name `normalObj.serial`, the same name as the expected one from my code.

Exploiting this Java insecure deserialization use case

We will now compile our code and execute it. For that, I will use a java docker image because I will need two separate machines simulating the attacker's and the victim's. You already have Docker installed in the free lab available for you. I recommend you connect to it and follow along. If you already have a lab of your own with Docker installed, that's great!

We will start a brand new container while mapping the `/tmp` directory in our host to the `/tmp` directory in the guest. That way, we can get the serialized file and use it later.

```
docker run -it -v /tmp:/tmp:rw java /bin/bash
```

When you get the prompt, download the `JavaSerial.java` code available in this gist. Then, we will compile the code and run it

```
javac JavaSerial.java && java JavaSerial && exit
```

Now we will spin up the victim's Docker container

```
docker run -v /tmp:/tmp:rw java /bin/bash
```

The, we will download the `JavaDeserial.java` file. Finally, we will compile and execute our deserialization operation

```
javac JavaDeserial.java && java JavaDeserial
```

Notice how we get the directory listing of the `/tmp` directory, meaning that the command `ls` has been executed even if the application didn't expect to use the `VulnObj` in the code at all.

If you find it difficult to understand the code, don't worry, there are only two things to remember from this example:

- The deserialization code has run our own `ls` command from the stored file without any verification even if the deserialized object is expected to be of `NormalObj` type.
- The second important thing to note is the magic number of the serialized object. Let's base64 encode the stored file and discover it.

```
cat vulnobj.serial | base64
#Result will be rO0ABXXXXXXXXXXXXXXXXXXXXXXXXXXX
```

The idea I want you to remember is whenever you encounter `rO0AB` in a base64 string in an application, think about testing it against the Java insecure deserialization vulnerability.

Insecure deserialization impact

As you saw so far, a successful attack leads to arbitrary code execution. This means that the impact will damage Confidentiality, Integrity and Availability. However, if the attacker can't build the right serialized payload, he can still trigger an exception and crash the server, therefore impacting Availability.

Insecure deserialization tools

There are many tools which can assist you when you are hunting for insecure deserialization vulnerabilities. In the case of Java, you can use the Java Deserialization Scanner Burp Suite extension. It allows you to test for different libraries using predefined POP gadget chains. Then, you can use Ysoserial to generate the appropriate payload. However, you noticed how these tools didn't help much in the previous insecure deserialization WebGoat challenge. Therefore, I think that the most valuable tool for this vulnerability would be a good deal of white-box testing, some patience and analysis skills.

Insecure deserialization attacks

Many attacks are exploiting this vulnerability in many different languages. We will explore some of them in this section.

Insecure deserialization attacks

Let's start with a PHP object injection example. In CVE-2018-20717, Prestashop suffered from a PHP insecure deserialization vulnerability. In this particular case, Prestashop uses user-controlled input in the unserialize function, which is a bad practice as we've seen earlier. However, notice how fixing a bad practice by implementing a filter is not a good idea. That CVE is exactly a demonstration of that!

In the world of Java, there is a classic example of Java insecure deserialization with the commons-collection library. Since so many Frameworks use this library, CVE-2015-7501 targets all of them at once. And the impact leads straight to remote code execution. There are even tools which exploit it on the fly.

Insecure deserialization Hackerone reports

In this insecure deserialization write up, you can learn the steps required to achieve a PHP object injection exploit. Notice how a successful attack generally involves going through a series of actions, reading the code and thinking outside the box. However, the general idea that we explained before remains the same and will act as a compass in your hacking journey.

In this insecure deserialization POC, you will find the steps which you can replicate when you want to exploit CVE-2015-7501 that we mentioned earlier using the Ysoserial tool. Specifically, it targets Jboss's JMXInvokerServlet vulnerable servlets.

Finally, this CTF write-up is a great exercise which combines many vulnerabilities, including two insecure deserialization vulnerabilities, to achieve remote code execution.

How to exploit insecure deserialization?

In this tutorial, we will exploit a Java insecure deserialization on OWASP WebGoat.

Insecure deserialization detection

A lot of people wonder how to detect insecure deserialization vulnerabilities in Java. Well, it's no magic. In our case, we will first perform a black-box approach. Therefore, we will explore the application until we find a payload which starts with `rO0AB` as we explained earlier. In challenge 5, under the Insecure deserialization menu, notice how the application expects a serialized java object.

How to test insecure deserialization ?

The easiest way to exploit it is to follow the instructions that we mentioned earlier in the Jboss Hackerone write-up. In other words, you generate a payload using the Ysoserial tool. Optionally, you can use the Java Deserialization Scanner Burp plugin to detect which library to build your payload on. This is well documented here. However, the latest release of WebGoat performs some checks before deserialization. In fact, In line 57 of the challenge source code file, WebGoat checks if the object to deserialize is an instance of the VulnerableTaskHolder class. Likely for us, this class performs OS commands in line 56. The `taskAction` attribute holds the command from the input which we control in the constructor.

```
public VulnerableTaskHolder(String taskName, String taskAction) {
        super();
        this.taskName = taskName;
        this.taskAction = taskAction;
        this.requestedExecutionTime = LocalDateTime.now();
}
```

Insecure deserialization : The VulnerableTaskHolder class accepts the taskAction in its constructor

Generate the payload

Now that we found the right class to serialize as well as where to inject our command. The goal is to cause a sleep of 5 seconds on the application. I've prepared the following gist which stores the serialized payload in a file on disk, exactly like we did in our previous Java insecure deserialization example.

I've downloaded the WebGoat release code. Then, I've used my IDE to create a Java class in the vulnerable package and pasted the gist. Finally, always from my IDE, I run the code which generates my `serial` file.

Exploiting the vulnerability

Now that we have the malicious payload, we simply need to base64 encode it and send it in the application's input field.

```
cat serial | base64 -w0
```

Notice how the payload starts with `rO0AB`.

```
rO0ABXNyADFvcmcuZHVtbXkuaW5zZWN1cmUuZnJhbWV3b3JrLlZ1bG5lcmFibGVUYXNrSG9sZGVyAAAA
aW11dAAZTGphdmEvdGltZS9Mb2NhbERhdGVUaW11O0wACnRhc2tBY3Rpb250ABJMamF2YS9sYW5nL1N0
YS50aW11LlNlcpVdhLobIkiyDAAAeHB3DgUAAAfkAhoVOBw4RUQMeHQAB3NsZWVwIDV0AAVkdW1teQ==
```

Insecure deserialization: payload ready to be used

Sending this payload will effectively cause a sleep of 5 seconds.

> Note: The vulnerable class checks if the token is older than 10 minutes. You need to generate one yourself and use it within 10 minutes.

Hopefully, this insecure deserialization tutorial gave you the roadmap of how to research and exploit this vulnerability. For more info about other languages, head to the OWASP insecure deserialization cheat sheet.

Insecure deserialization mitigation

As you might have concluded from what we saw, you should never trust data when you deserialize it. You perform checks on whitelisted classes you expect. This depends on each programming language. For example, Python provides you with the ability to restrict classes. For java, you saw how the WebGoat challenge checks if the serialized data is of type `VulnerableTaskHolder`.

If you'd want to implement solutions which don't depend on a language, think about using data formats like JSON or XML, and use digital signatures. You can find more on that in the OWASP insecure deserialization prevention cheat sheet.

Insecure deserialization references

- A must read for Java deserialization.
- Pickle documentation.
- PHP Object Injection.
- OWASP Insecure deserialization Cheat sheet.

Outro

If you are new to hacking and want to learn the basics, read the Ultimate Guide to OWASP Top 10, which covers both the theory and the hands-on exercises you'll need to kickstart your career. I have prepared a dedicated hacking lab that you can download and use for free. Many other articles on hacking and bug bounty hunting are available on thehackerish.com.

As usual, here is the video tutorial.

Previous Post:
Cross-site Scripting, XSS explained

Next Post:
Using components with known vulnerabilities

0 Comments thehackerish Disqus' Privacy Policy Login
Recommend Sort by Best
Start the discussion…
LOG IN WITH OR SIGN UP WITH DISQUS
Name
Be the first to comment.
Subscribe Add Disqus to your site Do Not Sell My Data DISQUS

Categories

Overcome your struggles and become a successful bug bounty hunter!

Get you Free hacking lab VM

free-hacking-lab-vm

© 2021 thehackerish