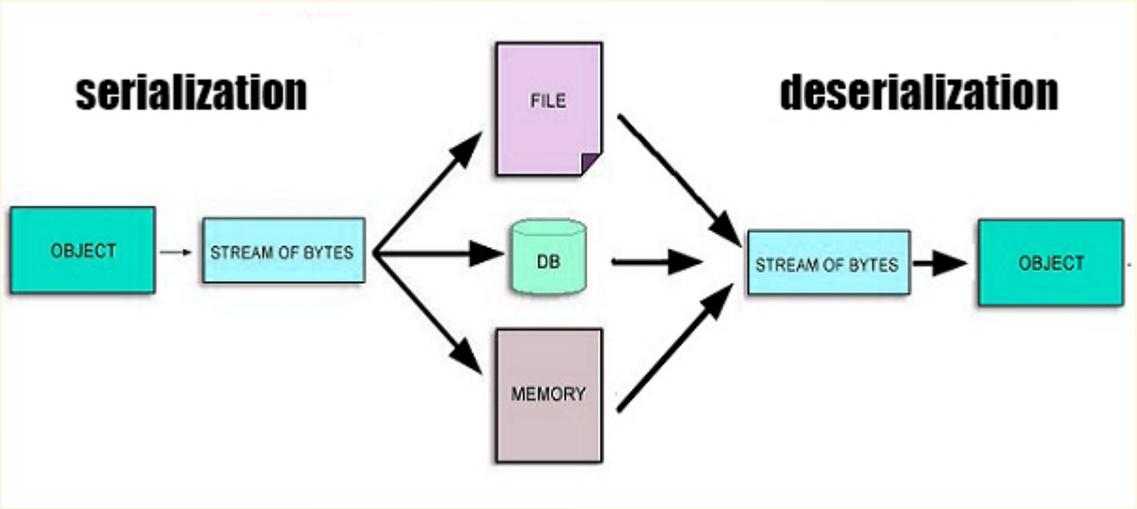
Deserialization

Serialization

> Object serialization transforms an object's data to a bytestream that represents the state of the data. The serialized form of the data contains enough information to recreate the object with its data in a similar state to what it was when saved. [^1]



Deserialization

```
InputStream is = request.getInputStream();
ObjectInputStream ois = new ObjectInputStream(is);
AcmeObject acme = (AcmeObject)ois.readObject();
```

- The casting operation to `AcmeObject` occurs **after** the deserialization process ends
- It is not useful in preventing any attacks that happen during deserialization from occurring

Insecure Deserialization

- Insecure deserialization often leads to **remote code execution** (RCE), one of the most serious attacks possible
- Other possible attacks include
  - replay attacks
  - injection attacks
  - privilege escalation
  - DoS

Risk Rating

Insecure Deserialization

Exploitability	Prevalence	Detecability	Impact	Risk
🔸 Difficult	🔶 Common	🔶 Average	🔴 Severe	A8
(**1**	+ **2**	+ **2**) / 3	* **3**	= **5.0**

Attack Example (Adobe BlazeDS)

```
[RemoteClass(alias="javax.swing.JFrame")]
public class JFrame {
    public var title:String = "Gotcha!";
    public var defaultCloseOperation:int = 3;
    public var visible:Boolean = true;
}
```

- Above payload creates a `JFrame` instance on the target server
- The `JFrame` object will have a `defaultCloseOperation` of value `3`
- This indicates that **the JVM should exit** when this window is closed

Exercise 6.1

1. What happens when the `root` object would be deserialized?

```
ArrayList<Object> root = new ArrayList<>(Integer.MAX_VALUE);
```

Exercise 6.2

1. What happens when the `root` object would be deserialized?

```
Set root = new HashSet();
Set s1 = root;
Set s2 = new HashSet();
for (int i = 0; i < 100; i++) {
  Set t1 = new HashSet();
  Set t2 = new HashSet();
  t1.add("foo");
  s1.add(t1);
  s1.add(t2);
  s2.add(t1);
  s2.add(t2);
  s1 = t1;
  s2 = t2;
}
```

Prevention

- **Avoid native deserialization formats 💯**
  - JSON/XML lessens (but not removes) the chance of custom deserialization logic being maliciously repurposed
- Use the Data Transfer Object (DTO) pattern
  - Exclusive purpose is data transfer between application layers

If serialization cannot be avoided

- Sign any serialized objects & only deserialize signed data
- Enforce strict type constraints during deserialization before object creation (Not sufficient on its own!)
- Isolate deserialization in low privilege environments
- Log deserialization exceptions and failures
- Restrict or monitor incoming and outgoing network connectivity from containers or servers that deserialize
- Monitor & alert if a user deserializes constantly

✓ SerialKiller (Java)

Replacing every `java.io.ObjectInputStream` instanciation

```
ObjectInputStream ois = new ObjectInputStream(is);
String msg = (String) ois.readObject();
```

with `SerialKiller` from a look-ahead Java deserialization library

```
ObjectInputStream ois = new SerialKiller(is, "/etc/serialkiller.conf");
String msg = (String) ois.readObject();
```

secures the application from untrusted input. Via `serialkiller.conf` classes can be black- or whitelisted.

❌ node-serialize (JavaScript)

The `node-serialize` module uses `eval()` internally for deserialization, allowing exploits like

```
var serialize = require('node-serialize');
var x = '{"rce":"_$$ND_FUNC$$_function (){console.log(\'exploited\')}()"}'
serialize.unserialize(x);
```

⚠️ *The affected version `0.0.4` of `node-serialize` is also the latest version of this module!*

Exercise 6.3

1. Find the „NextGen“ successor to the half-heartedly deprecated XML-based B2B API in the Juice Shop
2. Exploit this API with a DoS-like Remote Code Exeution

ℹ *If the server would need >2sec to process your attack request, it is considered „DoS-like“ enough.*