# Deserialization

**Deserialization attacks** occur when untrusted or tampered data is **deserialized (converted back to objects)** by an application without proper validation. If an attacker controls this input, they can craft **malicious serialized objects** that, when deserialized, **execute arbitrary code**, change logic, or corrupt data.

# What is Serialization/Deserialization?

| Term | Meaning |
| --- | --- |
| Serialization | Converting an object (e.g., user data, settings) into a byte stream (for storage or transmission). |
| Deserialization | Reconstructing the original object from this byte stream. |

# Impact

| Type | Description |
| --- | --- |
| Remote Code Execution (RCE) | Arbitrary system commands during deserialization |
| Access Control Bypass | Elevate user roles or change permissions |
| Denial of Service (DoS) | Application crash or resource exhaustion |
| Tampering with App Logic | Alter serialized data to change user roles or settings |
| Data Corruption | Invalid objects cause application malfunctions |

# Types of Deserialization Attacks

## 1. Insecure Deserialization with RCE

* **Definition**: Deserializing untrusted data triggers execution of dangerous code (via magic methods like \_\_wakeup() or readObject()).
* **Example**: Java deserializes attacker-supplied object with Runtime.exec() inside.
* **Mitigation**: Never deserialize untrusted input; use signed or encrypted objects.

## 🔹 2. Logic Manipulation (Non-RCE)

* **Definition**: Attacker alters serialized data (e.g., user role, token) to change app behavior.
* **Example**: user=admin changed from user=guest in serialized blob.
* **Mitigation**: Validate object state after deserialization, use HMACs or encryption.

## 🔹 3. Denial of Service

* **Definition**: Send recursive or memory-exhaustive objects to crash the app.
* **Example**: Python pickle object with circular references or deep nesting.
* **Mitigation**: Set limits on deserialization size and depth.

# Language-Specific Deserialization Risks

| Language | Risky Functions / Formats |
| --- | --- |
| Java | ObjectInputStream, Java Serialization API |
| PHP | unserialize(), \_\_wakeup(), \_\_destruct() |
| Python | pickle.loads(), marshal |
| .NET | BinaryFormatter, DataContractSerializer |
| Node.js | serialize-javascript, node-serialize |
| Ruby | Marshal.load() |

# Example (PHP unserialize RCE)

class Logger {

public $logFile;

function \_\_destruct() {

unlink($this->logFile); // dangerous

}

}

# Attacker creates:

O:6:"Logger":1:{s:7:"logFile";s:15:"/etc/passwd";}

# Result: unlink() called on sensitive file

# Testing Techniques

| Technique | Tool | Description |
| --- | --- | --- |
| Tamper serialized data | Burp Suite / Postman | Modify cookies, parameters |
| Use gadget chains | ysoserial / marshalsec | Inject serialized payloads that trigger RCE |
| Check error messages | Observe behavior after injecting junk data |  |
| Check app logic | Look for encoded values that hold roles, config |  |
| Inspect parameters | Cookies, JWTs, hidden fields with base64/hex data |  |

# Tools to Detect/Exploit

| **Tool** | **Use** |
| --- | --- |
| **ysoserial** (Java) | Generate malicious serialized objects for known gadget chains |
| **marshalsec** | Exploit Java deserialization (JNDI, LDAP) |
| **GadgetInspector** | Detect gadget chains in Java codebases |
| **PHPGGC** | Generate PHP gadget chains |
| **Burp Suite** | Manual injection, fuzzing, tampering |
| SerialSniper / FeroxDeser | Java and .NET deserialization testing |

# Mitigation:

| Category | Control |
| --- | --- |
| Avoid Deserialization of Untrusted Data | Do not accept serialized input from users |
| Use Safer Formats | Prefer JSON or XML with strict schemas over binary object formats |
| Implement Integrity Checks | Sign serialized objects (e.g., HMAC, JWT) |
| Use Whitelisting | Accept only specific, expected object types |
| Use Safe Libraries | Replace unserialize() with safe alternatives (e.g., json\_decode() in PHP) |
| Code Review | Look for magic methods and deserialization points (\_\_wakeup, \_\_destruct, etc.) |
| Input Size Limits | Limit request body and data depth |

Avoid deserializing untrusted data

Use safe serialization libraries

# Points

“Deserialization is dangerous **not because of deserialization itself**, but because the application trusts what gets reconstructed.”

“Check for insecure deserialization by injecting payloads in cookies, tokens, or parameters that look serialized (e.g., base64 blobs).”

“Avoid object-based session handling — prefer **stateless tokens like JWTs** or **HMAC-signed values**.”

# Real-World Vulnerabilities

| System | Details |
| --- | --- |
| Apache Commons Collections (Java) | Popular RCE gadget chain exploited via ObjectInputStream |
| Jenkins (2017) | Deserialization in remoting caused RCE |
| Ruby on Rails | Marshal.load() vulnerabilities |
| PHP 7.0/7.1 | Multiple unserialize-related CVEs |

# Common Patterns to Watch

| Pattern | Action |
| --- | --- |
| O: or s: in base64 data | PHP serialization |
| rO0A | Java serialization magic number (base64) |
| pickle.loads() | Python, often in ML/data APIs |
| eyJhbGciOi... | JWT – ensure alg is not none |