INSECURE

DESERIALIZATION

Complete Attack Reference

> **What is Insecure Deserialization?**
> Deserialization converts serialized data back into objects. When untrusted data is deserialized without validation, attackers can manipulate serialized objects to achieve Remote Code Execution (RCE), DoS, authentication bypass, or privilege escalation. Ranked #8 in OWASP Top 10 due to its critical impact.

1. Java Deserialization RCE

Description: Java ObjectInputStream deserializes untrusted data. Magic bytes AC ED 00 05 identify serialized Java objects. Libraries like Commons Collections, Spring, Apache have gadget chains. Ysoserial tool generates exploits. Leads to instant RCE.

Example Attack:

```
Cookie: serialized_user=rO0AB... (base64 encoded Java object with malicious payload)
```

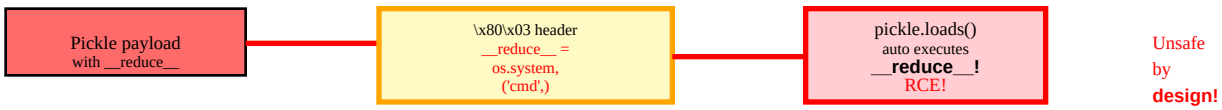
Attack Techniques:

```
ysoserial CommonsCollections6 'cmd' | Apache Commons, Spring gadget chains
```

> **Impact:** Remote Code Execution, full system compromise

2. Python Pickle RCE

Description: Pickle serializes Python objects. Unsafe by design - can execute arbitrary code during unpickling. __reduce__ method exploited for RCE. Never unpickle untrusted data. Used in web frameworks, ML model files, caching.



Example Attack:

```
import pickle; pickle.loads(malicious_data) executes attacker's code
```

Attack Techniques:

```
__reduce__ method with os.system | eval() execution | Command injection
```

Impact: Remote Code Execution, Python application compromise

3. PHP Object Injection

Description: PHP unserialize() on untrusted data. Magic methods (__wakeup, __destruct, __toString) auto-executed. POP (Property-Oriented Programming) chains exploit existing classes. Leads to RCE, SQL injection, file operations.

Example Attack:

```
unserialize($_COOKIE['data']) with O:4:"User":1:{s:4:"role";s:5:"admin";} payload
```

Attack Techniques:

```
Magic method chains | POP gadgets | PHPGGC tool for exploit generation
```

Impact: Remote Code Execution, authentication bypass, arbitrary file operations

4. .NET Deserialization

Description: .NET BinaryFormatter, DataContractSerializer vulnerable. TypeNameHandling in JSON.NET enables attacks. ViewState in ASP.NET if machine key compromised. YSoSerial.Net generates payloads. Full RCE on Windows servers.

Example Attack:

```
BinaryFormatter.Deserialize(stream) with malicious TypeConfuseDelegate payload
```

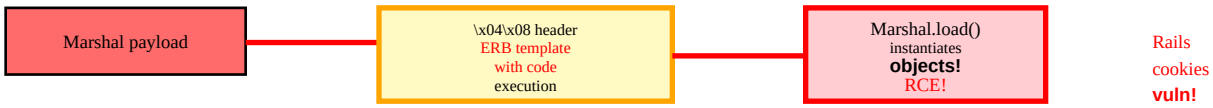
Attack Techniques:

```
YSoSerial.Net gadget chains | ViewState exploitation | TypeNameHandling abuse
```

Impact: Remote Code Execution on .NET/Windows applications

5. Ruby Marshal RCE

Description: Marshal loads/dumps Ruby objects. Unsafe for untrusted data. Can instantiate arbitrary objects. Rails applications often vulnerable. Cookie stores, caching, session data exploitable.



Example Attack:

```
Marshal.load(user_input) with crafted payload executes code
```

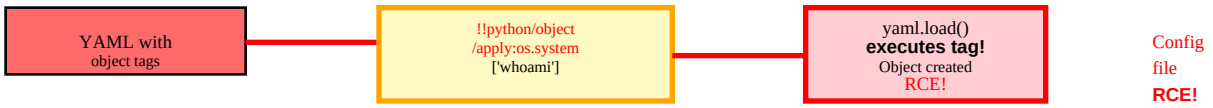
Attack Techniques:

```
ERB template injection | Gem-specific gadget chains | Object instantiation
```

Impact: Remote Code Execution on Ruby/Rails applications

6. YAML Deserialization

Description: YAML parsers (PyYAML, SnakeYAML) execute code. !!python/object tag creates objects. !!java/object in Java. Often in config files, APIs accepting YAML. RCE in Jenkins, many others.



Example Attack:

```
!!python/object/apply:os.system ['whoami'] executes command
```

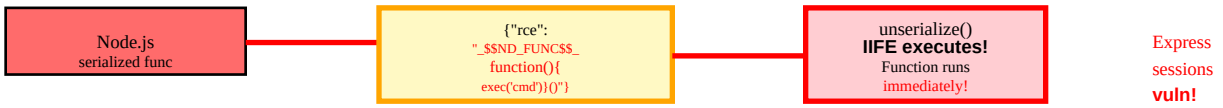
Attack Techniques:

```
!!python/object exploitation | !!java/object in Java | Tag abuse
```

Impact: Remote Code Execution, config file injection

7. Node.js Deserialization

Description: node-serialize, serialize-javascript vulnerable. IIFE (Immediately Invoked Function Expression) execution. JSON.parse with __proto__ pollution. Express session stores exploitable.



Example Attack:

```
{"rce":"_$$ND_FUNC$$_function(){require('child_process').exec('cmd')}()"}
```

Attack Techniques:

```
IIFE injection | Prototype pollution | Function serialization
```

> **Impact:** Remote Code Execution on Node.js applications

8. XML Deserialization (XStream)

Description: XStream library deserializes XML to Java objects. Dynamic proxy exploitation. ProcessBuilder chains for command execution. Used in RESTful services. Many public exploits available.

Example Attack:

```
XML containing ProcessBuilder chains deserializes to RCE
```

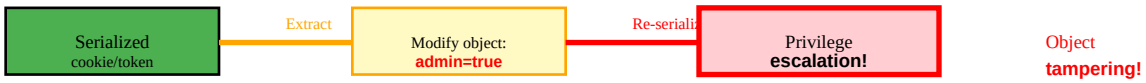
Attack Techniques:

```
ProcessBuilder gadget chains | Dynamic proxy exploitation | XStream CVEs
```

> **Impact:** Remote Code Execution via XML deserialization

9. Token/Cookie Tampering

Description: Serialized data in JWT, cookies, session tokens. Modify serialized object: change user role, bypass authentication. Sign with weak/known keys. Mass assignment vulnerabilities.



Example Attack:

```
Deserialize cookie, change 'admin':false to 'admin':true, re-serialize
```

Attack Techniques:

```
Object modification | Role manipulation | Privilege escalation via tampering
```

Impact: Authentication bypass, privilege escalation

10. ViewState Exploitation (ASP.NET)

Description: ASP.NET ViewState stores page state. Encrypted/signed with machine key. If machine key known/weak, inject malicious ViewState. YSoSerial.Net ViewStatePayloadGenerator creates exploits.

Example Attack:

```
__VIEWSTATE parameter with malicious serialized .NET object
```

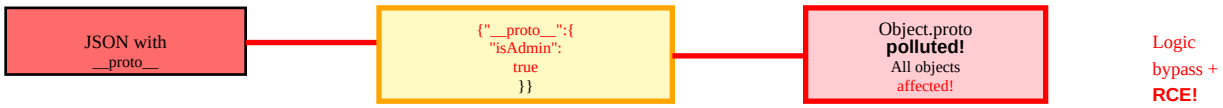
Attack Techniques:

```
Machine key extraction | ViewState deserialization gadgets | YSoSerial.Net
```

Impact: Remote Code Execution on ASP.NET applications

11. Prototype Pollution (JavaScript)

Description: Modify Object.prototype affects all objects. Merge functions vulnerable. __proto__ property manipulation. Leads to RCE in some contexts, DoS, authentication bypass.



Example Attack:

```
JSON.parse('{"__proto__":{"isAdmin":true}}') pollutes prototype
```

Attack Techniques:

```
__proto__ manipulation | constructor.prototype | Gadget chains
```

Impact: RCE in Node.js, client-side attacks, logic bypass

12. Java RMI/JMX Exploitation

Description: Java RMI (Remote Method Invocation) deserializes method arguments. JMX (Java Management Extensions) similar. Network-accessible Java RMI endpoints vulnerable. Metasploit modules available.

Example Attack:

```
RMI Registry on port 1099 accepts malicious serialized objects
```

Attack Techniques:

```
RMI exploitation | JMX connector abuse | Distributed Java attacks
```

Impact: Remote Code Execution on Java enterprise applications

IDENTIFYING SERIALIZED DATA

Format	Magic Bytes/Pattern	Common Location
Java	AC ED 00 05 (hex)	Cookies, RMI, JMX
Java (base64)	rO0AB... (starts)	HTTP parameters, headers
PHP	O:4:"User":... or a:2:{...}	Cookies, session data
Python Pickle	\x80\x03 or \x80\x04	Cached data, ML models
.NET Binary	00 01 00 00 00 FF FF...	ViewState, remoting
Node.js	{"rce":"_$$ND_FUNC$$_...	JSON with functions
Ruby Marshal	\x04\x08 (binary)	Rails cookies, cache
YAML	--- or !!python/object	Config files, APIs

EXPLOITATION TOOLS

Java Deserialization

• ysoserial - Generates Java deserialization payloads for multiple gadget chains

• Java Deserialization Scanner (Burp) - Automated detection extension

• marshalsec - Java unmarshaller security research tool

• SerializationDumper - Analyzes Java serialization streams

.NET Deserialization

• YSoSerial.Net - .NET deserialization payload generator

• ViewState decoder tools - Decode/analyze ASP.NET ViewState

• BinaryFormatter exploits - Pre-built .NET payloads

PHP Deserialization

• PHPGGC - PHP Generic Gadget Chains (POP chain generator)

• PHP Object Injection Scanner - Automated vulnerability detection

• Serialized Object Manipulator - Modify PHP serialized data

Python/Ruby

• Pickle Payloads - Python pickle RCE generators

• Marshal exploit scripts - Ruby Marshal attack tools

General Purpose

- Burp Suite - Proxy, modify serialized data
- CyberChef - Decode/encode various formats
- Metasploit - RMI, JMX exploitation modules

TESTING METHODOLOGY

1. Identify Serialized Data: Search for serialized data in: cookies, hidden form fields, API parameters, session tokens, cache entries. Look for magic bytes (AC ED, rO0, \x80\x03). Check Content-Type headers. Base64 decode suspicious parameters.

2. Determine Format: Identify serialization format: Java (AC ED 00 05), PHP (O:X:, a:X:), Python Pickle (\x80), .NET (binary formatter), Node.js (function serialization). Use detection tools and magic byte analysis.

3. Test for Deserialization: Modify serialized data and observe behavior. Change object properties, add new properties. Look for: error messages revealing class names, timing differences, exceptions. Confirms deserialization occurs.

4. Find Gadget Chains: Research application dependencies. Check for vulnerable libraries: Apache Commons Collections, Spring Framework, FastJSON. Use ysoserial/PHPGGC to find applicable gadgets. Version-specific exploits common.

5. Generate Payload: Use appropriate tool: ysoserial (Java), YSoSerial.Net (.NET), PHPGGC (PHP). Specify gadget chain and command. Encode payload correctly (base64, URL encoding). Test locally first if possible.

6. Deliver Payload: Inject malicious serialized object in identified location. Use Burp Suite to modify requests. Try multiple injection points. Monitor for out-of-band callbacks (DNS, HTTP). Watch for error messages.

7. Verify Exploitation: Confirm RCE with: DNS exfiltration (nslookup, dig), HTTP callback (curl, wget), file creation (touch, echo), sleep commands (timing). Use safe commands first. Escalate to reverse shell.

8. Test Edge Cases: Try different: gadget chains, payload encodings, injection points, serialization versions. Test with compressed data. Check for WAF bypasses. Time-based blind exploitation.

PREVENTION & MITIGATION

• **Never Deserialize Untrusted Data:** PRIMARY RULE: Don't deserialize user-controlled input. If you must, use safe formats like JSON (without special parsing). Sign/encrypt serialized data with strong keys. Implement strict input validation.

• **Use Safe Serialization Formats:** Prefer JSON over binary serialization. Use simple data structures only. Avoid formats that execute code (Pickle, YAML with tags). No object instantiation from untrusted sources.

• **Implement Integrity Checks:** Sign serialized data with HMAC. Verify signature before deserialization. Use strong cryptographic keys. Rotate keys regularly. Store keys securely (environment variables, key vaults).

• **Apply Whitelisting:** Whitelist allowed classes for deserialization. Java: implement ObjectInputStream with resolveClass override. .NET: use SerializationBinder. PHP: implement __wakeup validation. Deny by default.

• **Update Dependencies:** Keep serialization libraries updated. Remove vulnerable libraries if unused. Apache Commons Collections versions with gadget chains. Regular security audits of dependencies.

• **Isolate Deserialization:** Run deserialization in sandboxed environment. Limit file system and network access. Use containers/VMs for isolation. Apply principle of least privilege. Monitor for suspicious activity.

• **Monitor and Log:** Log all deserialization operations. Alert on exceptions/errors. Monitor for: known gadget class names, unusual object types, execution attempts. Implement runtime application self-protection (RASP).

• **Code Review:** Review all deserialization code paths. Check session management. Audit cookie handling. Search codebase for: unserialize, pickle.loads, ObjectInputStream, BinaryFormatter. Security testing in CI/CD.

DETECTION CHECKLIST

✓ Intercept all HTTP traffic with Burp Suite

✓ Search for base64 encoded data in cookies, parameters, headers

✓ Decode base64 and check for magic bytes (AC ED, \x80\x03, etc.)

✓ Look for serialization format indicators (O:X:, a:X:, rO0)

✓ Identify application framework (.NET, Java, PHP, Python, Node.js)

✓ Research framework-specific deserialization vulnerabilities

✓ Test for deserialization by modifying serialized data

✓ Check error messages for class names and stack traces

✓ Use ysoserial/PHPGGC/YSoSerial.Net to generate test payloads

✓ Test with DNS callback payloads (Burp Collaborator)

✓ Try multiple gadget chains for same framework

✓ Test all identified serialization points systematically

✓ Check ViewState in ASP.NET applications

✓ Test JWT tokens for deserialization issues

✓ Look for Java RMI services (port 1099)

✓ Search for YAML configuration endpoints

✓ Test file upload with serialized objects

✓ Check for prototype pollution in JavaScript applications

POPULAR GADGET CHAINS

Chain Name	Target Library	Capability
CommonsCollections1-7	Apache Commons Collections	Java RCE
Spring1-2	Spring Framework	Java RCE
Rome	ROME library	Java RCE
JSON1	Jackson, FastJSON	Java RCE
Hibernate1-2	Hibernate ORM	Java RCE
TypeConfuseDelegate	.NET Framework	.NET RCE
ObjectDataProvider	.NET Framework	.NET RCE

Guzzle/RCE	Guzzle (PHP)	PHP RCE
Monolog/RCE	Monolog (PHP)	PHP RCE
Symfony/RCE	Symfony (PHP)	PHP RCE

Note: This cheat sheet is for educational and authorized security testing only. Unauthorized exploitation of deserialization vulnerabilities is illegal. Always obtain written permission before testing.