# Preventing Deserialization Vulnerabilities

## Introduction

With vulnerability assessment and exploit development covered, let's look at deserialization from a defender/developers point of view, and discuss what can be done to prevent deserialization vulnerabilities from occurring.

## Guidelines

#### 1. Avoid Deserializing User Input

The most effective way to prevent deserialization vulnerabilities from being exploited, is to never deserialize user-input. If an attacker can not control the serialized input, then no payload can be passed to the deserialization method.

#### 2. Avoid Unnecessary Deserialization

Sometimes it is not necessary to use serialization to store data. For example, the "Remember Me" token in TeeTrove could have easily been a JWT or just plain JSON, neither of which would have required deserialization.

#### 3. Use Secure Serialization Mechanisms

Avoid using serialization mechanisms such as BinaryFormatter which are known to have issues. For .NET, Microsoft [recommends](https://learn.microsoft.com/en-us/dotnet/standard/serialization/binaryformatter-security-guide#preferred-alternatives) using the following serializers:

* [XmlSerializer](https://learn.microsoft.com/en-us/dotnet/api/system.xml.serialization.xmlserializer?view=net-7.0) for XML
* [DataContractSerializer](https://learn.microsoft.com/en-us/dotnet/api/system.runtime.serialization.datacontractserializer?view=net-7.0) for XML
* [BinaryReader](https://learn.microsoft.com/en-us/dotnet/api/system.io.binaryreader?view=net-7.0) and [BinaryWriter](https://learn.microsoft.com/en-us/dotnet/api/system.io.binarywriter?view=net-7.0) for XML and JSON
* [System.Text.JSON](https://learn.microsoft.com/en-us/dotnet/api/system.text.json?view=net-7.0) for JSON

However, relying solely on these classes does not guarantee the total elimination of deserialization vulnerabilities. For instance, the second vulnerability in TeeTrove involved abusing XmlSerializer.

#### 4. Use Explicit Types

Many serializers in .NET allow developers to specify explicit types during deserialization, which prevents objects of other types from being parsed. For example, with XmlSerializer an object type must be passed in the constructor. Unless the user can control this type (like in TeeTrove), it will not be possible to deserialize objects of any other type.

Code: csharp

XmlSerializer xs = new XmlSerializer(typeof(Person));  
Person p = (Person)xs.Deserialize(...);

#### 5. Use Signed Data

Cryptographically signing serialized data that users can modify is a robust defensive mechanism to hinder exploitation. For example, the authentication cookie used by TeeTrove was signed with a secret key, and if we did not have access to the source code, we would not have been able to generate a valid cookie that would be deserialized.

When choosing an algorithm to sign the data, it is important to consider that some algorithms are more secure than others. For example, a simple MD5 hash could be relatively simply brute-forced.

#### 6. Least Possible Privileges

Finally, running web servers while adhering to the Principle of Least Privileges (PoLP) is a recommended defensive security best practice. For an attacker exploiting a deserialization vulnerability and landing a reverse shell on a web server, implementing PoLP could mean the difference between causing limited damage (for example, leaking a database) or a catastrophic one (compromising the entire Active Directory domain).