# SER04-J. Do not allow serialization and deserialization to bypass the security manager

Serialization and deserialization features can be exploited to bypass security manager checks. A serializable class may contain security manager checks in its constructors for various reasons, including preventing untrusted code from modifying the internal state of the class. Such security manager checks must be replicated wherever a class instance can be constructed. For example, if a class enables a caller to retrieve sensitive internal state contingent upon security checks, those checks must be replicated during deserialization to ensure that an attacker cannot extract sensitive information by deserializing the object.

## Noncompliant Code Example

In this noncompliant code example, security manager checks are used within the constructor but are omitted from the `writeObject()` and `readObject(` methods that are used in the serialization-deserialization process. This omission allows untrusted code to maliciously create instances of the class.

```
public final class Hometown implements Serializable {
  // Private internal state
  private String town;
  private static final String UNKNOWN = "UNKNOWN";

  void performSecurityManagerCheck() throws AccessDeniedException {
    // ...
  }

  void validateInput(String newCC) throws InvalidInputException {
    // ...
  }

  public Hometown() {
    performSecurityManagerCheck();

    // Initialize town to default value
    town = UNKNOWN;
  }

  // Allows callers to retrieve internal state
  String getValue() {
    performSecurityManagerCheck();
    return town;
  }

  // Allows callers to modify (private) internal state
  public void changeTown(String newTown) {
    if (town.equals(newTown)) {
      // No change
      return;
    } else {
      performSecurityManagerCheck();
      validateInput(newTown);
      town = newTown;
    }
  }

  private void writeObject(ObjectOutputStream out) throws IOException {
    out.writeObject(town);
  }

  private void readObject(ObjectInputStream in) throws IOException {
    in.defaultReadObject();
    // If the deserialized name does not match the default value normally
    // created at construction time, duplicate the checks
    if (!UNKNOWN.equals(town)) {
      validateInput(town);
    }
  }
}
```

(Although there are security manager checks, the data in this example is not sensitive. Serializing unencrypted sensitive data violates SER03-J. Do not serialize unencrypted sensitive data.)

`AccessDeniedException` and `InvalidInputException` are both security exceptions that can be thrown by any method without requiring a `throws` declaration.

## Compliant Solution

This compliant solution implements the required security manager checks in all constructors and methods that can either modify or retrieve internal state. Consequently, an attacker cannot create a modified instance of the object (using deserialization) or read the serialized byte stream to reveal serialized data.

```
public final class Hometown implements Serializable {
  // ... All methods the same except the following:

  // writeObject() correctly enforces checks during serialization
  private void writeObject(ObjectOutputStream out) throws IOException {
    performSecurityManagerCheck();
    out.writeObject(town);
  }

  // readObject() correctly enforces checks during deserialization
  private void readObject(ObjectInputStream in) throws IOException {
    in.defaultReadObject();
    // If the deserialized name does not match the default value normally
    // created at construction time, duplicate the checks
    if (!UNKNOWN.equals(town)) {
      performSecurityManagerCheck();
      validateInput(town);
    }
  }
}
```

Refer to SEC04-J. Protect sensitive operations with security manager checks for information about implementing the `performSecurityManagerCheck(` method, which is important for protection against finalizer attacks.

The `ObjectInputStream.defaultReadObject()` fills the object's fields with data from the input stream. Because each field is deserialized recursively, it is possible for the `this` reference to escape from control of the deserialization routines. This can happen if a referenced object publishes the `this` reference in its constructors or field initializers (see TSM01-J. Do not let the this reference escape during object construction for more information). To be compliant, recursively deserialized subobjects must not publish the `this` object reference.

## Risk Assessment

Allowing serialization or deserialization to bypass the security manager may result in classes being constructed without required security checks.

| Rule | Severity | Likelihood | Remediation Cost | Priority | Level |
|---|---|---|---|---|---|
| SER04-J | High | Probable | High | P6 | L2 |

## Automated Detection

| Tool | Version | Checker | Description |
|---|---|---|---|
| Parasoft Jtest | 10.3 | **SECURITY.WSC.SCSER** | Implemented |

## Related Guidelines

| | |
|---|---|
| Secure Coding Guidelines for Java SE, Version 5.0 | Guideline 8-4 / SERIAL-4: Duplicate the SecurityManager checks enforced in a class during serialization and deserialization |

## Android Implementation Details

The `java.security` package exists on Android for compatibility purposes only, and it should not be used.

## Bibliography

| | |
|---|---|
| [Long 2005] | Section 2.4, "Serialization" |