OpenJDK
Installing
Contributing
Sponsoring
Developers' Guide
Vulnerabilities
JDK GA/EA Builds
Mailing lists
Wiki · IRC
Bylaws · Census
Legal
Workshop
JEP Process
Source code
Mercurial
GitHub
Tools
Git
jtreg harness
Groups
(overview)
Adoption
Build
Client Libraries
Compatibility & Specification Review
Compiler
Conformance
Core Libraries
Governing Board
HotSpot
IDE Tooling & Support
Internationalization
JMX
Members
Networking
Porters
Quality
Security
Serviceability
Vulnerability
Web
Projects
(overview, archive)
Amber
Audio Engine
CRaC
Caciocavallo
Closures
Code Tools
Coin
Common VM Interface
Compiler Grammar
Detroit
Developers' Guide
Device I/O
Duke
Font Scaler
Galahad
Graal
Graphics Rasterizer
IcedTea
JDK 7
JDK 8
JDK 8 Updates
JDK 9
JDK (..., 21, 22)
JDK Updates
JavaDoc.Next
Jigsaw
Kona
Kulla
Lambda
Lanai
Leyden
Lilliput
Locale Enhancement
Loom
Memory Model Update
Metropolis
Mission Control
Modules
Multi-Language VM
Nashorn
New I/O
OpenJFX
Panama
Penrose
Port: AArch32
Port: AArch64
Port: BSD
Port: Haiku
Port: Mac OS X
Port: MIPS
Port: Mobile
Port: PowerPC/AIX
Port: RISC-V
Port: s390x
Portola
SCTP
Shenandoah
Skara
Sumatra
Tiered Attribution
Tsan
Type Annotations
Valhalla
Verona
VisualVM
Wakefield
Zero
ZGC
ORACLE

JEP 290: Filter Incoming Serialization Data

Owner	Roger Riggs
Type	Feature
Scope	SE
Status	Closed / Delivered
Release	9
Component	core-libs / java.io:serialization
Discussion	core dash libs dash dev at openjdk dot java dot net
Effort	S
Duration	S
Relates to	JEP 415: Context-Specific Deserialization Filters
Reviewed by	Alan Bateman, Andrew Gross, Brian Goetz
Endorsed by	Brian Goetz
Created	2016/04/22 16:06
Updated	2022/08/15 16:17
Issue	8154961

Summary

Allow incoming streams of object-serialization data to be filtered in order to improve both security and robustness.

Goals

- Provide a flexible mechanism to narrow the classes that can be deserialized from any class available to an application down to a context-appropriate set of classes.
- Provide metrics to the filter for graph size and complexity during deserialization to validate normal graph behaviors.
- Provide a mechanism for RMI-exported objects to validate the classes expected in invocations.
- The filter mechanism must not require subclassing or modification to existing subclasses of ObjectInputStream.
- Define a global filter that can be configured by properties or a configuration file.

Non-Goals

- Define or maintain any specific policy of what classes should be allowed or disallowed.
- Fix complexity or integrity issues with specific classes or implementations.
- Provide look-ahead capabilities in the stream.
- Provide fine-grained visibility into the contents of objects.

Success Metrics

- Minimal measurable performance impact for simple reject-listing of classes

Motivation

Security guidelines consistently require that input from external sources be validated before use. The filter mechanism will allow object-serialization clients to more easily validate their inputs, and exported RMI objects to validate invocation arguments.

Description

The core mechanism is a filter interface implemented by serialization clients and set on an `ObjectInputStream`. The filter interface methods are called during the deserialization process to validate the classes being deserialized, the sizes of arrays being created, and metrics describing stream length, stream depth, and number of references as the stream is being decoded. The filter returns a status to accept, reject, or leave the status undecided.

For each new object in the stream the filter is called, with the class of the object, before the object is instantiated and deserialized. The filter is not called for primitives or `java.lang.String` instances that are encoded concretely in the stream. For each array, regardless of whether it is an array of primitives, array of strings, or array of objects the filter is called with the array class and the array length. For each reference to an object already read from the stream, the filter is called so it can check the depth, number of references, and stream length. Filter actions are logged to the `java.io.serialization` logger, if logging is enabled.

For RMI, the object is exported via a `UnicastServerRef` that sets the filter on the `MarshalInputStream` to validate the invocation arguments as they are unmarshalled. Exporting objects via UnicastRemoteObject should support setting a filter to be used for unmarshalling.

Process-wide Filter

A process-wide filter is configured via a system property or a configuration file. The system property, if supplied, supersedes the security property value.

- System property `jdk.serialFilter`
- Security property `jdk.serialFilter` in `conf/security/java.security`

A filter is configured as a sequence of patterns, each pattern is either matched against the name of a class in the stream or a limit. Patterns are separated by ";" (semi-colon). Whitespace is significant and is considered part of the pattern.

A limit pattern contains a "=" and sets a limit. If a limit appears more than once the last value is used. If any of the values in the call to `ObjectInputFilter.checkInput(...)` exceeds the respective limit, the filter returns Status.REJECTED. Limits are checked before classes regardless of the order in the sequence of patterns.

- `maxdepth=value` — the maximum depth of a graph
- `maxrefs=value` — the maximum number of internal references
- `maxbytes=value` — the maximum number of bytes in the input stream
- `maxarray=value` — the maximum array size allowed

Other patterns, from left to right, match the class or package name as returned from `Class::getName`. If the class is an array type, the class or package to be matched is the element type. Arrays for any number of dimensions are treated the same as the element type. For example, a pattern of "`!example.Foo`", rejects creation of any instance or array of `example.Foo`.

- If the pattern starts with "`!`", the class is rejected if the rest of the pattern matches, otherwise it is accepted
- If the pattern contains "/", the non-empty prefix up to the "/" is the module name. If the module name matches the module name of the class then the remaining pattern is matched with the class name. If there is no "/", the module name is not compared.
- If the pattern ends with "`.**`" it matches any class in the package and all subpackages
- If the pattern ends with "`.*`" it matches any class in the package
- If the pattern ends with "`*`", it matches any class with the pattern as a prefix.
- If the pattern is equal to the class name, it matches.
- Otherwise, the status is undecided.

ObjectInputFilter Interface and API

The object-input filter interface is implemented by clients of RMI and serialization, and provides the behaviors of the process-wide configurable filter.

```
interface ObjectInputFilter {
    Status checkInput(FilterInput filterInfo);

    enum Status {
        UNDECIDED,
        ALLOWED,
        REJECTED;
    }

   interface FilterInfo {
         Class<?> serialClass();
         long arrayLength();
         long depth();
         long references();
         long streamBytes();
   }

    public static class Config {
        public static void setSerialFilter(ObjectInputFilter filter);
        public static ObjectInputFilter getSerialFilter(ObjectInputFilter filter) ;
        public static ObjectInputFilter createFilter(String patterns);
    }
}
```

ObjectInputStream Filter

`ObjectInputStream` has additional methods to set and get the current filter. If no filter is set for an `ObjectInputStream` then the global filter is used, if any.

```
public class ObjectInputStream ... {
    public final void setObjectInputFilter(ObjectInputFilter filter);
    public final ObjectInputFilter getObjectInputFilter(ObjectInputFilter filter);
}
```

Alternatives

Modify existing subclasses and methods, but that would require changes that would inhibit use in third party implementations.

Testing

No existing tests need to be updated. New unit tests will test the filter mechanisms with serialized streams, RMI exported objects, and the global filtering mechanism.

Risks and Assumptions

The metrics presented to the filter supporting reject lists, accept lists, and stream metrics should be sufficient. When applied to the known use cases, some additional filter mechanisms may be discovered.

New APIs and interfaces will be introduced for JDK 9. Backporting this feature to previous versions will require the introduction of implementation-specific APIs to avoid changes to older versions of the Java SE specification.