Mitigating Java Deserialization attacks from within the JVM

Apostolos Giannakidis
@cyberApostle

BSides Luxembourg
20th October 2017

Who is

BACKGROUND

- Security Architect at Waratek
- AppSec
- Runtime protection
- Vulnerability and exploit analysis
- R&D exploit mitigation
- MSc Computer Science

Key Takeaways

- Black/White listing is not an enterprise-scale solution
- Instrumentation Agents can be manipulated
- Runtime Virtualization offers privilege separation and memory isolation
- Runtime Privilege De-escalation safeguards application components and the JVM from API Abuse

Why should I care?

Attack Vectors

- RPC/IPC
- Message Brokers
- Caching
- Tokens / Cookies
- RMI
- JMX
- JMS
- …

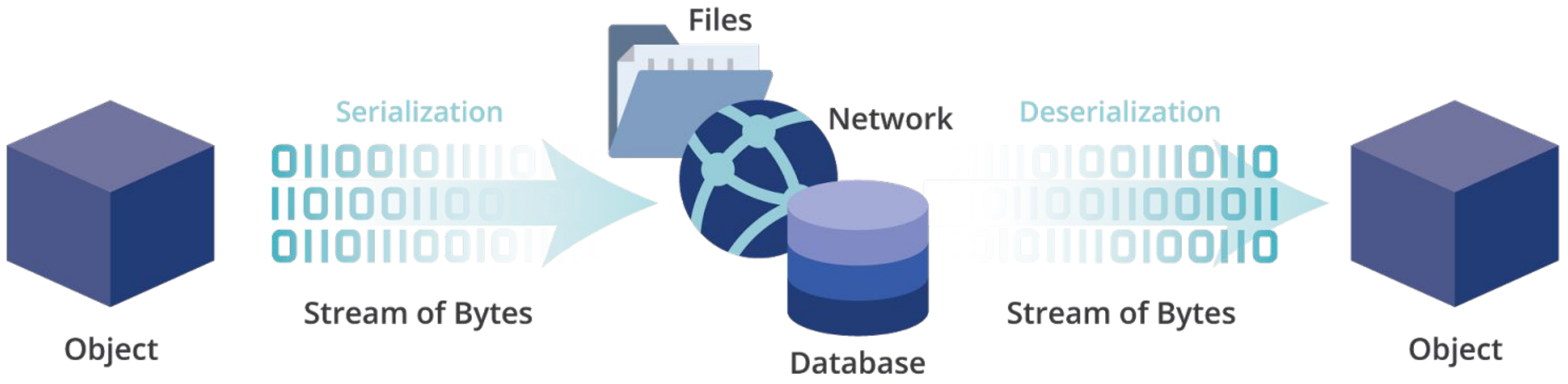
Attack Surface

- Oracle
- Red Hat
- Apache
- IBM
- Symantec
- Cisco
- Atlassian
- Adobe
- …

Impact & Popularity

- Reliable attack
- Easy to exploit
- All software layers
- OWASP Top 10 2017
- Oracle Java SE CPUs
- SF Muni breach
  - ~ 900 computers
  - ~$560k daily loss

Serialization 101



Deserialization of untrusted data

```
InputStream untrusted = request.getInputStream();
ObjectInputStream ois = new ObjectInputStream(untrusted);
SomeObject deserialized = (SomeObject) ois.readObject();
```

What is the problem here?

Deserialization of untrusted data

```
InputStream **untrusted** = request.getInputStream();
ObjectInputStream ois = new ObjectInputStream(untrusted);
SomeObject deserialized = (SomeObject) ois.**readObject()**;
```

What is the problem here?

- Any available class can be deserialized
- Deserializing untrusted data can result in malicious behavior
  - Arbitrary code execution
  - Denial of Service
  - Remote command execution
    - Malware / Ransomware infection

How to solve the problem?

- Stop using deserialization
  - Requires significant refactoring
  - Requires architectural changes
  - Endpoints in other software layers?
  - Legacy software?
- Patch your software
  - Could break the application
  - “*It is possible that some REST actions stop working*” - CVE-2017-9805
  - Oracle CPU October 2017 breaks backwards compatibility

Existing Runtime Mitigation Techniques

Java Security Manager

- Custom Security Policy

Filtering Class names

- Serialization Filtering (JEP-290)
- Custom Instrumentation Agents

Runtime Virtualization

- Micro-compartmentalization
- Privilege De-escalation

Java Security Manager

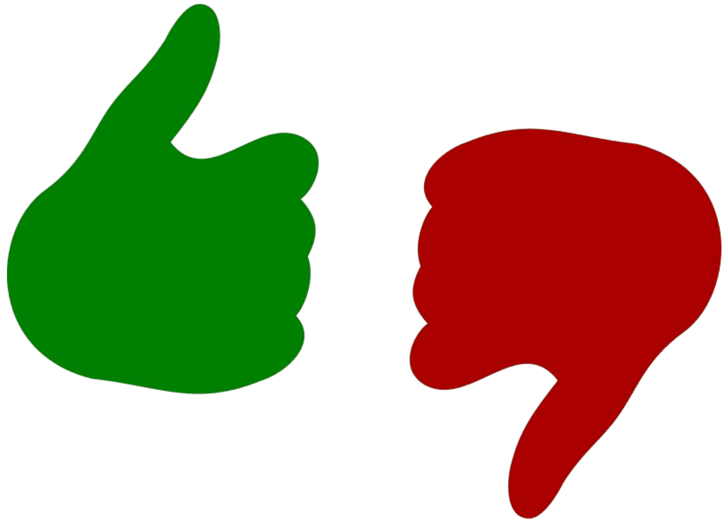
- Difficult to configure it correctly
- Performance issues
- No protection against DoS attacks
- No protection against deferred deserialization attacks

Critical Patch Update	**# vuls that can bypass the sandbox**
October 2017	**18**
July 2017	**26**
April 2017	**8**
January 2017	**14**

Discussion Time: Filtering class names

Blacklisting

Whitelisting



Discussion Time: Filtering class names

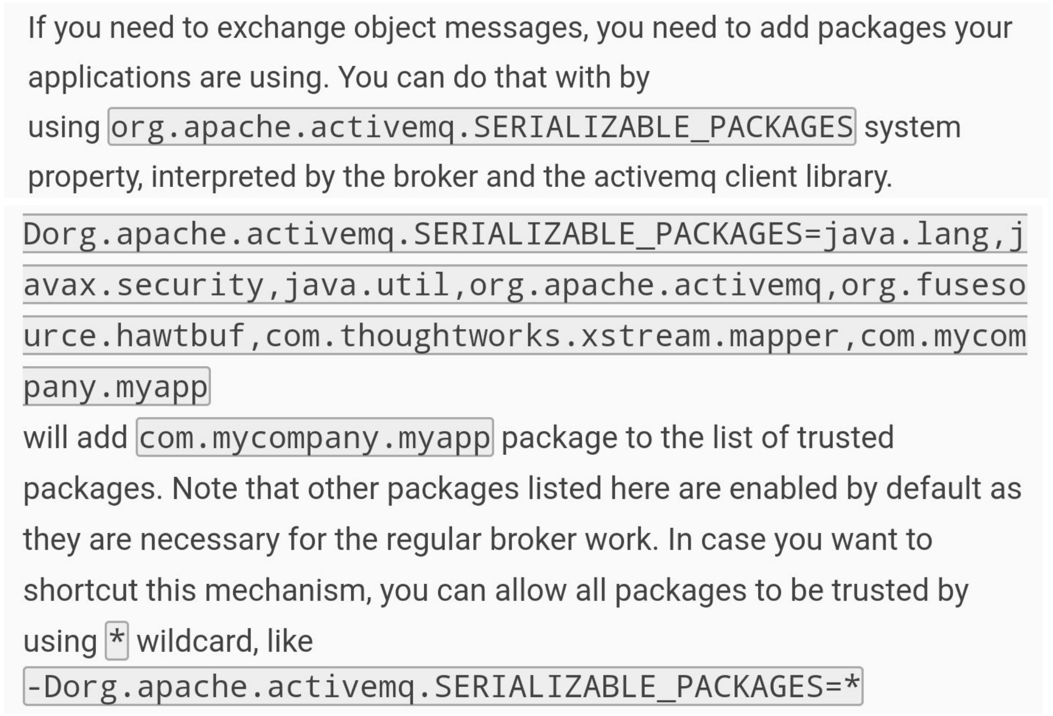
Blacklisting

- Requires profiling
- Never complete
- False sense of security
- Not possible if class is needed
- Can be bypassed

Whitelisting

- Requires profiling
- Difficult to do it right
- False positives if misconfigured
- No protection if class is needed
- No protection against Golden Gadgets
- Requires code reviews & testing

Whitelists are commonly mistreated



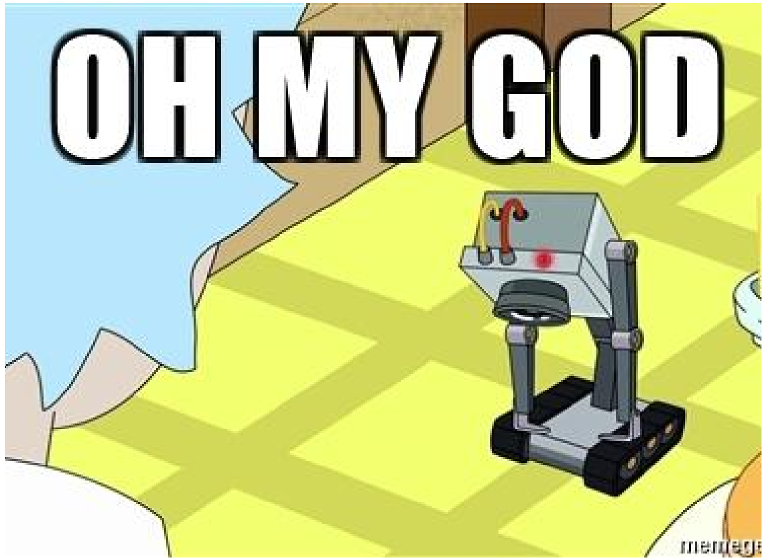
If you need to exchange object messages, you need to add packages your applications are using. You can do that with by using `org.apache.activemq.SERIALIZABLE_PACKAGES` system property, interpreted by the broker and the activemq client library.

```
Dorg.apache.activemq.SERIALIZABLE_PACKAGES=java.lang,j
avax.security,java.util,org.apache.activemq,org.fuseso
urce.hawtbuf,com.thoughtworks.xstream.mapper,com.mycom
pany.myapp
```

will add `com.mycompany.myapp` package to the list of trusted packages. Note that other packages listed here are enabled by default as they are necessary for the regular broker work. In case you want to shortcut this mechanism, you can allow all packages to be trusted by using `*` wildcard, like `-Dorg.apache.activemq.SERIALIZABLE_PACKAGES=*`

http://activemq.apache.org/objectmessage.html

Maintaining lists is a shity job

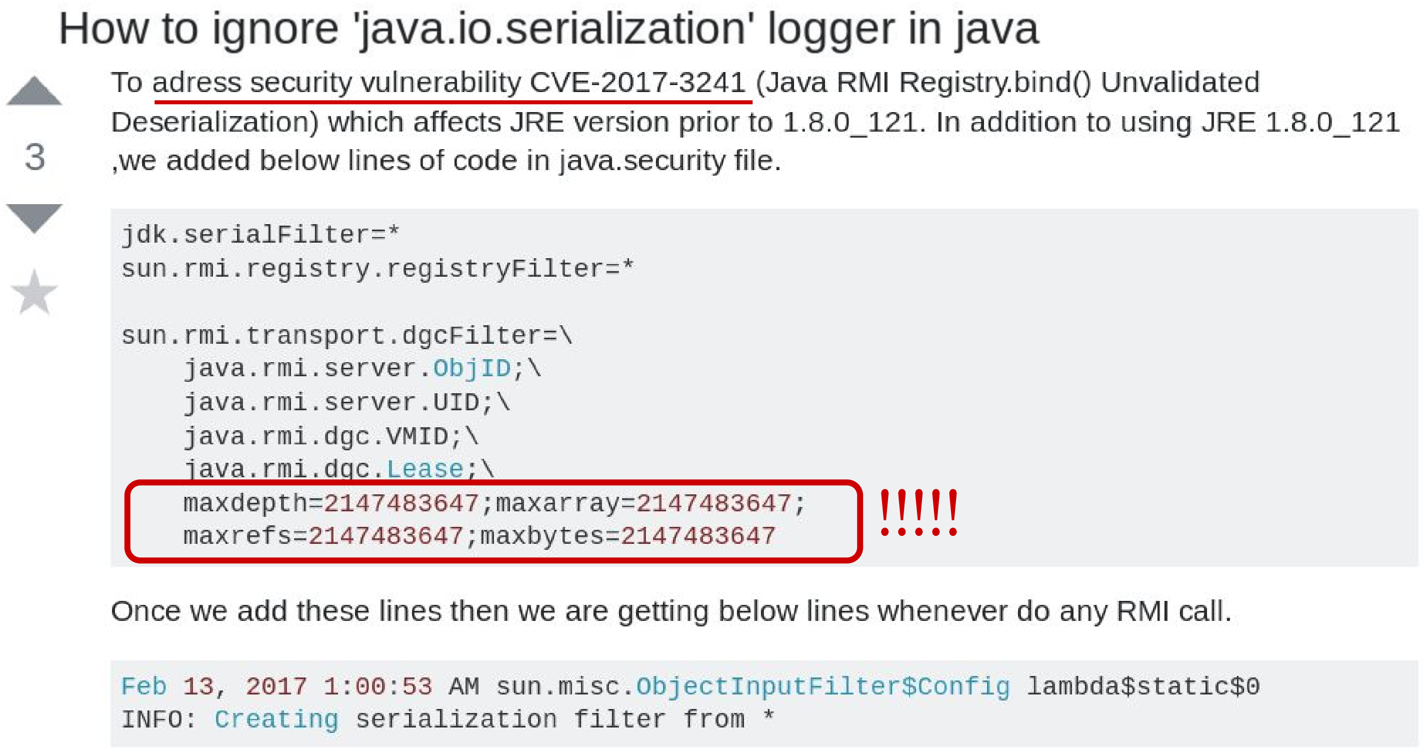


Serialization Filtering (JEP-290)

- Introduced in Java 9 on January 2017
  - Backported to Java 6, 7 and 8
  - But not available in older JVM versions (e.g. 7u21)
- White / Black listing approach
- 3 types of filters
  - Global Filter, Custom Filters, Built-in Filters
- Graph and Stream Limits
  - Requires knowledge of graphs, JVM internals and details of all deployed code
  - Easy to get them wrong

Serialization Filtering problems

https://stackoverflow.com/questions/42364744/how-to-ignore-java-io-serialization-logger-in-java



How to ignore 'java.io.serialization' logger in java

3

To adress security vulnerability CVE-2017-3241 (Java RMI Registry.bind() Unvalidated Deserialization) which affects JRE version prior to 1.8.0_121. In addition to using JRE 1.8.0_121 ,we added below lines of code in java.security file.

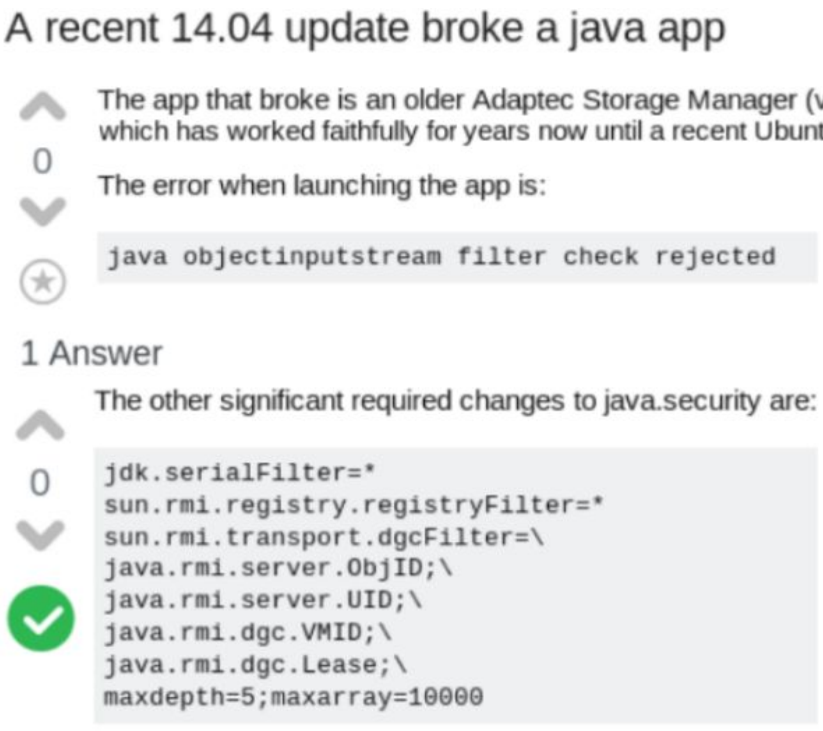
```
jdk.serialFilter=*
sun.rmi.registry.registryFilter=*

sun.rmi.transport.dgcFilter=\
    java.rmi.server.ObjID;\
    java.rmi.server.UID;\
    java.rmi.dgc.VMID;\
    java.rmi.dgc.Lease;\
    maxdepth=2147483647;maxarray=2147483647;
    maxrefs=2147483647;maxbytes=2147483647
```

!!!!!

Once we add these lines then we are getting below lines whenever do any RMI call.

```
Feb 13, 2017 1:00:53 AM sun.misc.ObjectInputFilter$Config lambda$static$0
INFO: Creating serialization filter from *
```

Serialization Filtering problems



A recent 14.04 update broke a java app

0

The app that broke is an older Adaptec Storage Manager (v4.30), which has worked faithfully for years now until a recent Ubuntu 14.04 update.

The error when launching the app is:

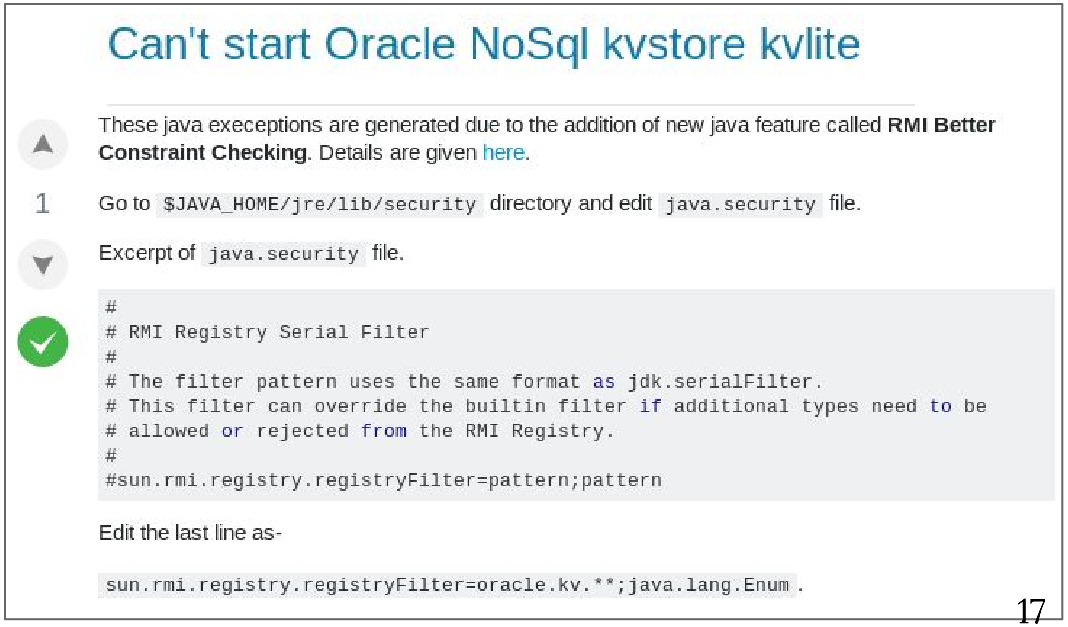
```
java objectinputstream filter check rejected
```

1 Answer

0

The other significant required changes to java.security are:

```
jdk.serialFilter=*
sun.rmi.registry.registryFilter=*
sun.rmi.transport.dgcFilter=\
java.rmi.server.ObjID;\
java.rmi.server.UID;\
java.rmi.dgc.VMID;\
java.rmi.dgc.Lease;\
maxdepth=5;maxarray=10000
```



Can't start Oracle NoSql kvstore kvlite

1

These java execeptions are generated due to the addition of new java feature called **RMI Better Constraint Checking**. Details are given here.

Go to `$JAVA_HOME/jre/lib/security` directory and edit `java.security` file.

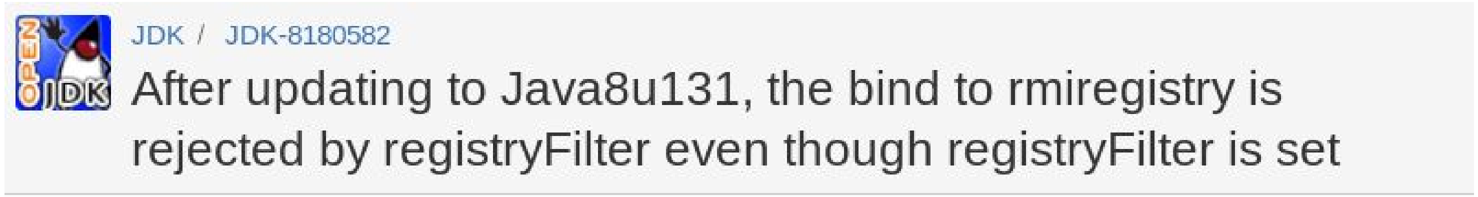
Excerpt of `java.security` file.

```
#
# RMI Registry Serial Filter
#
# The filter pattern uses the same format as jdk.serialFilter.
# This filter can override the builtin filter if additional types need to be
# allowed or rejected from the RMI Registry.
#
#sun.rmi.registry.registryFilter=pattern;pattern
```

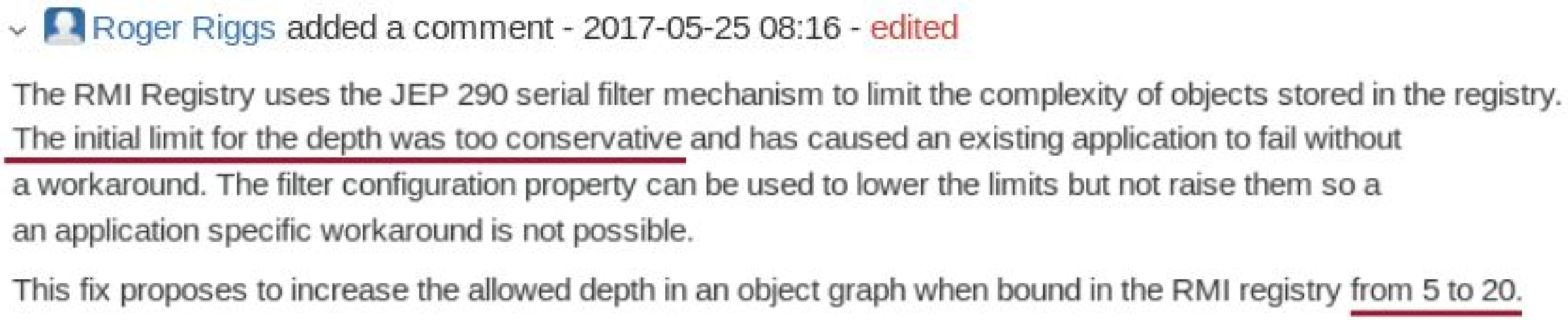
Edit the last line as-

`sun.rmi.registry.registryFilter=oracle.kv.**;java.lang.Enum`.

Serialization Filtering problems


JDK / JDK-8180582

After updating to Java8u131, the bind to rmiregistry is rejected by registryFilter even though registryFilter is set


Roger Riggs added a comment - 2017-05-25 08:16 - edited

The RMI Registry uses the JEP 290 serial filter mechanism to limit the complexity of objects stored in the registry. The initial limit for the depth was too conservative and has caused an existing application to fail without a workaround. The filter configuration property can be used to lower the limits but not raise them so a an application specific workaround is not possible.

This fix proposes to increase the allowed depth in an object graph when bound in the RMI registry from 5 to 20.

Instrumentation Agents

- Instrumentation API
- Black/ White listing approach
  - Global Filter
  - Custom Filters
- Known open source agents
  - NotSoSerial
  - Contrast-rO0
  - more…

?

What is the problem with Instrumentation Agents?

What is the problem with Instrumentation Agents?

- Instrumentation API was **not** designed for Security

  From the Javadoc API:

> *Instrumentation is the addition of byte-codes to methods* ***for the purpose of gathering data****.*
>
> *Since the changes are purely additive, these tools* ***do not modify application state or behavior****.*
>
> *Examples of such* ***benign*** *tools include monitoring agents, profilers, coverage analyzers, and event loggers.*

https://docs.oracle.com/javase/8/docs/api/java/lang/instrument/Instrumentation.html

Single Fault Domain

- Instr. agents and application share the same address space
- No separation of privileges
- Nothing prevents an app exploit to modify agent code/data
- Think of the browser/plugin, kernel/user-space paradigm
- Agents can be compromised by application attack vectors
- No protection against insider attacks
- Inappropriate for Cloud environments

Instrumentation Agents can turn into Double Agents

- *Reporting* & *Blacklisting* mode not suitable for production
- Configuration tampering at runtime
  - Backdoor deployment
  - Agent becomes DoS attack vector
- Protection can be disabled
- Log entries cannot be trusted

Demo PoC: Turn Contrast-rOo against itself

Setup

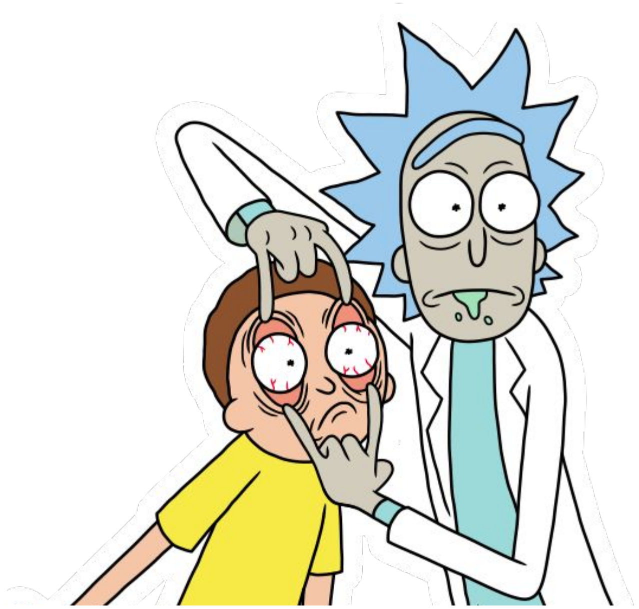
- Deploy the Contrast-rO0 instrumentation agent
- Use the default configuration file
- Run Tomcat with a vulnerable sample app

Goal

- Tamper agent's runtime configuration
- Remove blacklisted classes (aka add backdoors)

Source: https://github.com/maestros/fileuploadapp

Let’s study the attack carefully



```
ObjectInputStream.readObject()
    AnnotationInvocationHandler.readObject()
        Map(Proxy).entrySet()
            AnnotationInvocationHandler.invoke()
                LazyMap.get()
                    ...
                        InvokerTransformer.transform()
                            Method.invoke()
                                Runtime.exec()
```

Source: Chris Frohoff
Marshalling Pickles
AppSecCali 2015

```
LinkedHashSet.readObject()
  ...
    LinkedHashSet.add()
      ...
        Proxy(Templates).equals()
          ...
            ClassLoader.defineClass()
                    Class.newInstance()
                      ...
                        Runtime.exec()
```

Source: Chris Frohoff
ysoserial

Let's revisit the core of the problem

- The JVM is *irrationally* too permissive
- The JVM makes no effort to mitigate API Abuse attacks
- It is not even safeguarding its own invariants!
- All code and data can be accessible from any context
    - without a Security Manager

What do the standards suggest?

CERT Secure Coding Standards

- SER08-J. Minimize privileges before deserializing from a privileged context
- SEC58-J. Deserialization methods should not perform potentially dangerous operations

MITRE

- CWE-250: Execution with Unnecessary Privileges
  - [...] isolate the privileged code as much as possible from other code. Raise privileges as late as possible, and drop them as soon as possible.
- CWE-273: Improper Check for Dropped Privileges
  - Compartmentalize the system to have "safe" areas where trust boundaries can be unambiguously drawn.

Runtime Virtualization Deserialization Mitigation

- Runtime Virtualization
  - Places security controls in an isolated address space
  - Offers complete visibility of all executed instructions
- Runtime Micro-compartmentalization
  - Defines boundaries around operations
  - Controlled communication between compartments
- Runtime Privilege De-escalation
  - Allows only non-privileged operations after each boundary
  - Safeguards JVM’s state
  - Protects against API abuse cases

Conclusion

- Maintaining lists does not scale and is a burden
- Filtering classes can be too low level for AppSec teams
- Instrumentation Agents can become Double Secret Agents
- Do not use agent’s Reporting & Blacklist mode in production
- The runtime platform must:
  - be secure-by-default
  - safeguard the developer’s code from being abused

Thanks!

Apostolos Giannakidis
@cyberApostle

BSides Luxembourg
20th October 2017

Discussion Time

- Bug hunting - Code reviewing deserialization gadgets
- Global Filters - Good or Bad?
- Attack detection using WAFs

Code Review #1

```
public class LookupTable implements Serializable {
	private transient TableElement[] lookupTable;
	public LookupTable(int size) {
		int elements = Math.min(Math.max(4,size),32);
		lookupTable = new TableElement[elements];
	}
	private void readObject(ObjectInputStream s)
		  throws IOException, ClassNotFoundException {
		int numEntries = s.readInt();
		lookupTable = new TableElement[numEntries];
	}
}
```

Code Review #2

```
public final class TempFile implements Serializable {
        private String fileName;
        private void readObject(ObjectInputStream s) {
              s.defaultReadObject();   // read the field
        }
        public String toString() {
              return fileName != null ? fileName : "";
        }
        private void finalize() {
              new File(fileName).delete();
        }
        public File getTempFile() {
              return new File(fileName);
        }
}
```

Know what you need to protect

Audit
Serializable classes

Create
Threat Model

Re-evaluate
Threat Model when
class evolves

Identify all
deserialization
end-points

Add authentication
in each end-point

Risk-based Management using lists

- Who should be responsible for their maintenance?
- Difficult to apply risk-based management
    - How should a class's risk profile be assessed?
    - Developers understand code
    - AppSec teams understand operations
        - OS, File System, Network, Database, etc.

What is the problem with Global Filters?

- A Global Filter is … Global (Process-wide)
- Applies to all deserialization endpoints
  - Even if the endpoint deserializes internal data
- Whitelist must include classes from all software layers
  - How do you know what classes are needed by each layer?
- Whitelisting with Global Filter increases risk exposure
  - Global Filter defines your deserialization attack surface

Check WAFs for False Positives

```
HashMap<String, String> map = new HashMap<>();
map.put(
“org.apache.commons.collections.functors.InvokerTransformer”,
             “calc.exe” );
FileOutputStream file = new FileOutputStream( "out.bin" );
ObjectOutputStream out = new ObjectOutputStream( file );
out.writeObject( map );
out.close();
```

```
java.lang.reflect.Field configField = ClassLoader.getSystemClassLoader()
        .loadClass("com.contrastsecurity.rO0.RO0Agent").getField("config");
Object configObj = configField.get(null);
Class<?> configClass = configObj.getClass();
java.lang.reflect.Field blacklistEnabledField =
        configClass.getDeclaredField("blacklistEnabled");
blacklistEnabledField.setAccessible(true);
blacklistEnabledField.setBoolean(configObj, false);
java.lang.reflect.Field blacklistField =
        configClass.getDeclaredField("blacklist");
blacklistField.setAccessible(true);
blacklistField.set(configObj, null);
```