Java deserialization vulnerability in QRadar RemoteJavaScript Servlet

From: "Securify B.V. via Fulldisclosure" <fulldisclosure () seclists org>
Date: Sat, 10 Oct 2020 14:03:18 +0200

```
--------------------------------------------------------------------------------
Java deserialization vulnerability in QRadar RemoteJavaScript Servlet
--------------------------------------------------------------------------------
Abstract
--------------------------------------------------------------------------------
A Java deserialization vulnerability exists in the QRadar
RemoteJavaScript Servlet. An authenticated user can call one of the
vulnerable methods and cause the Servlet to deserialize arbitrary
objects.

An attacker can exploit this vulnerability by creating a specially
crafted (serialized) object, which amongst other things can result in a
denial of service, change of system settings, or execution of arbitrary
code.

--------------------------------------------------------------------------------
See also
--------------------------------------------------------------------------------
CVE-2020-4280 [2]
6344079 [3] - IBM QRadar SIEM is vulnerable to deserialization of
untrusted data

--------------------------------------------------------------------------------
Tested versions
--------------------------------------------------------------------------------
This issue was successfully verified on QRadar Community Edition [4]
version 7.3.1.6 (7.3.1 Build 20180723171558).

--------------------------------------------------------------------------------
Fix
--------------------------------------------------------------------------------
IBM has released the following versions of QRader in which this issue
has been resolved:

- QRadar / QRM / QVM / QRIF / QNI 7.4.1 Patch 1 [5]
- QRadar / QRM / QVM / QRIF / QNI 7.3.3 Patch 5 [6]

--------------------------------------------------------------------------------
Introduction
--------------------------------------------------------------------------------
QRadar [7] is IBM's enterprise SIEM [8] solution. A free version of
QRadar is available that is known as QRadar Community Edition [4]. This
version is limited to 50 events per second and 5,000 network flows a
minute, supports apps, but is based on a smaller footprint for
non-enterprise use.

A Java deserialization vulnerability [9] exists in the QRadar
RemoteJavaScript Servlet. This Servlet contains a custom JSON-RPC [10]
implementation (based on JSON-RPC version 1.0). Certain methods accept
base64 encoded serialized Java objects. No checks have been implemented
to prevent deserialization of arbitrary objects. Consequently, an
authenticated user can call one of the affected methods and cause the
RemoteJavaScript Servlet to deserialize arbitrary objects.

An attacker can exploit this vulnerability by creating a specially
crafted (serialized) object,  which amongst other things can result in a
denial of service, change of system settings, or execution of arbitrary
code.

--------------------------------------------------------------------------------
Details
--------------------------------------------------------------------------------
The RemoteJavaScript Servlet is only accessible for authenticated users.
It is mapped to the following URLs:

- /remoteJavaScript
- /remoteMethod
- /JSON-RPC
- /JSON-RPC/*

The JSON data can be passed via the URL query string or as POST data.
The JSON data should contain a field named method, which contains the
name of the application and the method that needs to be invoked. The
requested application is looked up in the Application Registry. Each
application has a mapping XML file located under
/opt/qradar/conf/appconfig/ named <appname>-exported_methods.xml, which
is essentially a list of all (Java) methods that can be called including
their associated Java class, access control, and other settings.

When the application is found (and licensed), a call is made to
getExportedMethod() to lookup the Java method that needs to be invoked.
After some additional checks - like authorization - the Servlet will
eventually invoke the call() method of the found Java method. If
present, arguments are passed as a String array to the call() method.
These arguments are then converted into the correct type using the
com.qllabs.core.shared.util.ReflectionUtils.stringsToObjects() method.

com.qllabs.uiframeworks.application.ExportedMethod:
public abstract class ExportedMethod extends AllowableObject {

[...]

    public Object call(PageContext pageContext, String... passedArguments)
throws Exception {
        if (passedArguments != null && passedArguments.length != 0) {
            if (this.log.isDebugEnabled()) {
                this.log.debug("Calling with passed in arguments: " +
Arrays.toString(passedArguments));
            }

            return this.call(pageContext,
this.stringsToObjects(passedArguments));

[...]

    private Object[] stringsToObjects(String[] paramaters) throws
ExportedMethodException {
        return ReflectionUtils.stringsToObjects(this.getParameterTypes(),
paramaters);
    }

The parameter types differ per method and are provided via the
getParameterTypes() method. If the parameter type is a 'simple' type, it
```

```
will be converted without deserialization. However, some methods also
accept complex types, which are passed as serialized objects.

com.qllabs.core.shared.util.ReflectionUtils:
public class ReflectionUtils {
    public static Object stringToObject(Class type, String param) throws
NoSuchMethodException, InvocationTargetException,
InstantiationException, IllegalAccessException {
        long i;
        int i;
        if (type.isPrimitive()) {
            if (type.equals(Integer.TYPE)) {
                if (param == null) {
                    param = "0";
                }

                i = (new Double(param)).longValue();
                if (i > 2147483647L) {
                    i = 0L - (2147483647L - i);
                }

                return (int)i;
            } else if (type.equals(Character.TYPE)) {
                return param.charAt(0);
            } else if (type.equals(Double.TYPE)) {
                return new Double(param);
[...]
            } else {
                throw new RuntimeException("Unknown primitive type: " +
type.getName());
            }
        } else if (param != null && !param.equalsIgnoreCase("$_NULL_$")) {
            if (type.equals(Short.class)) {
                i = (new Double(param)).intValue();
                if (i > 32767) {
                    i = 0 - (32767 - i);
                }

                return (short)i;
[...]
            } else {
                return SerializationUtils.deserialize(Base64.decode(param));
            }
[...]
    }

    public static Object[] stringsToObjects(Class[] types, String...
parameters) throws RuntimeException {
        Pattern arrayMatcher = Pattern.compile("^\\[(.+)\\]$");
        if (parameters != null && parameters.length > 0) {
            Object[] newArgs = new Object[types.length];

            for(int i = 0; i < types.length; ++i) {
            try {
                Class type = types[ i ];
                if (!type.isArray()) {
                    newArgs[ i ] = stringToObject(type, parameters[ i ]);
[...]

Deserialization of objects is done using the
org.apache.commons.lang3.SerializationUtils class. This class doesn't
perform any checks on the objects that are deserialized. Since no checks
are done in the RemoteJavaScript Servlet it can be abused to deserialize
arbitrary objects. An attacker can exploit this vulnerability by
creating a specially crafted (serialized) object.

--------------------------------------------------------------------------
Proof of concept
--------------------------------------------------------------------------
The JSON-RPC interface already contains a method that allows running of
arbitrary commands (as the nobody user). This method is named
qradar.executeCommand and can be called by any user, no special
privileges are required. However, the method checks if the property
console.enableExecuteCommand exists and is set to true. By default this
property doesn't exist and thus it is not possible to call this method
to run arbitrary commands. By utilizing the deserialization
vulnerability it is possible to create this property, after which it is
possible to use qradar.executeCommand to run arbitrary commands.

com.qllabs.qradar.ui.qradarservices.UIQRadarServices:
public static Object executeCommand(PageContext pageContext, String
command, int timeoutSeconds) throws Exception {
    if
(!"true".equalsIgnoreCase(QSystem.getProperty("console.enableExecuteCommand")))
{
        throw new Exception("Cannot execute remote system commands");
    } else {
        File qradarDir = new File(NVAReader.getProperty("NVA",
"/opt/qradar"));
        Process proc = Runtime.getRuntime().exec(new String[]{"/bin/sh",
"-c",
command}, (String[])null, qradarDir);
[...]

One of the methods that can be used to trigger a deserialization
operation is the method qradar.validateChangesAssetConfiguration. This
method is mapped to the Java method
com.qllabs.assetprofilerconfiguration.ui.util.AssetProfilerConfig.validateChangesAssetConfiguration().
The method takes one argument of the type java.util.List.

The proof of concept uses a Jython gadget. The Jython Java library is
present in the Servlet's class path and consequently we can deserialize
objects found in this library. The ysoserial [11] payload generation
tool already contains a gadget [12] that uses the Jython library.
ysoserial's payload will first write a Python file to the target system,
after which the file is executed. The payload has been modified to
directly create the target property (console.enableExecuteCommand)
without the need to write a file to disk first. The payload is modified
to execute the following Python code (upon deserialization):

eval("__import__('com.qllabs.frameworks.util.QSystem', globals(),
locals(), ['setProperty'],
0).setProperty('console.enableExecuteCommand', 'true')")

https://gist.github.com/ykoster/90d3d13fe70c357ae93f5ddb3faee4f2

--------------------------------------------------------------------------
References
--------------------------------------------------------------------------
[1]
https://www.securify.nl/advisory/java-deserialization-vulnerability-in-qradar-remotejavascript-servlet
[2] https://cve.mitre.org/cgi-bin/cvename.cgi?name=CVE-2020-4280
[3] https://www.ibm.com/support/pages/node/6344079
[4] https://developer.ibm.com/qradar/ce/
[5]
https://www.ibm.com/support/fixcentral/swg/downloadFixes?
parent=IBM%20Security&product=ibm/Other+software/IBM+Security+QRadar+SIEM&release=7.4.0&platform=Linux&function=fixId&
fixids=7.4.1-QRADAR-QRSIEM-20200915010309&includeRequisites=1&includeSupersedes=0&downloadMethod=http&login=true
[6]
https://www.ibm.com/support/fixcentral/swg/downloadFixes?
parent=IBM%20Security&product=ibm/Other+software/IBM+Security+QRadar+Vulnerability+Manager&release=All&platform=All&fu
nction=fixId&fixids=7.3.3-QRADAR-QRSIEM-
20200929154613&includeRequisites=1&includeSupersedes=0&downloadMethod=http&login=true
[7] https://www.ibm.com/security/security-intelligence/qradar
[8] https://en.wikipedia.org/wiki/Security_information_and_event_management
[9] https://www.owasp.org/index.php/Top_10-2017_A8-Insecure_Deserialization
[10] https://en.wikipedia.org/wiki/JSON-RPC
[11] https://github.com/frohoff/ysoserial
[12]
https://github.com/frohoff/ysoserial/blob/master/src/main/java/ysoserial/payloads/Jython1.java
```

Current thread:

Java deserialization vulnerability in QRadar RemoteJavaScript Servlet ***Securify B.V. via Fulldisclosure (Oct 16)***