Vulnerability assessment of the VRSpace application

Audited for OpenComputing Ltd
Audited by Mikrotron d.o.o

20210703

Table of contents

[1]	Introduction	2
[2]	Scope	2
[3]	Methodology	2
[4]	About Us	2
[5]	Executive Summary	3
[6]	Code analysis	4
[6.1]	SAST	4
[6.2]	SonarQube	4
[6.3]	Semgrep	5
[6.4]	EnumJavaLibs	5
[6.5]	jQAssistant	6
[6.6]	Gadget inspector	6
[7]	IAST	6
[7.1]	Contrast Community Edition	7
[8]	DAST	8
[8.1]	OWASP Zed Attack Proxy (ZAP)	8
[8.2]	Burp Scan	9
[8.3]	Nikto	9
[8.4]	Wapiti	10
[9]	Manual code review and custom tooling	11
[9.1]	Custom fuzzers	11
[10]	Stress testing	14
[10.1]	Wsstat	14
[10.2]	Curl.sh	14
[11]	Dependency Check	15

[1] Introduction

This document reports the findings of a security assessment targeting VRSpace application, carried out by Mikrotron during May and June of 2021.

[2] Scope

- Source code available on Github [https://github.com/jalmasi/vrspace)
- Locally hosted web application

[3] Methodology

Our audit techniques included manual and tool assisted code analysis, user interface interaction as well as dynamic application security testing and fuzzing.

[4] About Us

Among the other things we do, we offer development services (Java applications and Arduino projects), installation and system administration of Linux servers and PostgreSQL DBMS, IT security testing, 3D print workshops, 3D print services, 3D modeling and many other. Feel free to contact us about it at mikrotron@mikrotron.hr.

We have many years of experience in these fields, but always want to learn something new - our services can go way beyond the list above.

[5] Executive Summary

Identified Vulnerabilities:

- **Denial of service through resource exhaustion [MEDIUM]**
  It is possible to use all websockets from a single source.

- **Components with known vulnerabilities [LOW]**
  Application uses several older version of dependencies that have since been updated due to vulnerabilities found in them:
  - jquery version 3.3.1.min
  - bcprov-jdk15on-1.64
  - neo4j-java-driver-4.0.2
  - Spring-core-5.2.14

- **Parameter injection [LOW]**
  It is possible to inject parameter pair [“parameter”:”value”] that will be forwarded to other clients within the same world.

[6] Code analysis

As a part of our engagement, we performed both tool assisted and manual code analysis.

[6.1] SAST

Static application security testing, also known as “white-box” testing, utilizes tools, such as static code analyzers, to scan application’s code in a non-running state, before the code is compiled.

Tool assisted code analysis was conducted using the following static application security testing tools:

- AppThreat sast-scan[1]
- AppThreat dep-scan[2]
- gSAST[3]
- semgrep[4]
- SonarQube[5]

While sast-scan, dep-scan and gSAST showed a few false positives, SonarQube and semgrep did not find any security vulnerabilities.

[6.2] SonarQube

SonarQube is an open-source platform for continuous inspection of code quality to perform automatic reviews with static analysis of code to detect bugs and security vulnerabilities.

SonarQube did not find any security vulnerabilities.

1 https://github.com/AppThreat/sast-scan
2 https://github.com/AppThreat/dep-scan
3 https://github.com/b0n1t0/gSAST
4 https://github.com/returntocorp/semgrep
5 https://www.sonarqube.org

[6.3] Semgrep

Semgrep is a fast, open-source, static analysis tool for finding bugs and enforcing code standards at editor, commit, and CI time.

```
techie@lubuntu:~/vrspace$ semgrep --config=p/ci
using config from https://semgrep.dev/p/ci. Visit https://semgrep.dev/registry to see all public rules.
downloading config...
running 135 rules...
100%|                                                                                                    |135/135
ran 135 rules on 87 files: 0 findings
techie@lubuntu:~/vrspace$ semgrep  --config=p/security-audit
using config from https://semgrep.dev/p/security-audit. Visit https://semgrep.dev/registry to see all public
rules.
downloading config...
running 245 rules...
100%|                                                                                                    |245/245
ran 245 rules on 657 files: 0 findings
```

Semgrep did not find any security vulnerabilities.

Since standard SAST tools did not find any security vulnerabilities, our next approach was aimed at the jar file itself.
For that purpose we used the following tools:

- jQAssistant[6]
- gadgetinspector[7]
- EnumJavaLibs[8]

As the application relies on Jackson to deserialize user's input, thorough inspection of Jackson implementation and potential deserialization vulnerabilities was conducted utilizing tools such as EnumJavaLibs and gadgetinspector, following with custom fuzzer and manual code review.

[6.4] EnumJavaLibs

EnumJavaLibs can be used to discover which libraries are loaded (i.e. available on the classpath) by a Java application when it supports deserialization.

```
techie@lubuntu:~/EnumJavaLibs/bin$ java -jar EnumJavaLibs.jar base64
[*] Fetching jars from /home/techie/.serially/jars..
[*] Serializing classes from 1 jars..
[*] Finished
[*] See output in /home/techie/EnumJavaLibs/bin/enumjavalibs_20210703173325.csv
```

6 https://jqassistant.org
7 https://github.com/JackOfMostTrades/gadgetinspector
8 https://github.com/redtimmy/EnumJavaLibs

[6.5] jQAssistant

While jQAssistant is a primarily QA tool, one of the use cases for the tool is to validate dependencies between modules of a project. It was used complimentary to manual code review in order to map relationships and dependencies within the project.

[6.6] Gadget inspector

This project inspects Java libraries and classpaths for gadget chains. Gadgets chains are used to construct exploits for deserialization vulnerabilities. By automatically discovering possible gadgets chains in an application's classpath penetration testers can quickly construct exploits and application security engineers can assess the impact of a deserialization vulnerability and prioritize its remediation.

Gadget inspector did not find any gadget chains, as shown in the screenshot below.

```
2021-06-30 12:04:50,295 gadgetinspector.GadgetInspector [INFO] Discovering gadget chain source methods...
2021-06-30 12:04:50,973 gadgetinspector.GadgetInspector [INFO] Searching call graph for gadget chains...
2021-06-30 12:04:53,768 gadgetinspector.GadgetChainDiscovery [INFO] Iteration 0, Search space: 819
2021-06-30 12:04:53,858 gadgetinspector.GadgetChainDiscovery [INFO] Iteration 1000, Search space: 732
2021-06-30 12:04:53,877 gadgetinspector.GadgetChainDiscovery [INFO] Iteration 2000, Search space: 254
2021-06-30 12:04:53,903 gadgetinspector.GadgetChainDiscovery [INFO] Found 0 gadget chains.
2021-06-30 12:04:53,908 gadgetinspector.GadgetInspector [INFO] Analysis complete!
```

[7] IAST

Interactive application security testing is a methodology of application testing where code is analyzed for security vulnerabilities while an application is running. IAST tools deploy agents and sensors in applications to detect issues in real-time during a test.

The application can be run by an automated test or by a human tester to find vulnerabilities in the application.

[7.1] Contrast Community Edition

Contrast Community Edition[9] for Java focuses on software security and delivers real-time security telemetry from within running code.

As shown in the screenshot above, Contrast CE did not find any security vulnerabilities.

9 https://www.contrastsecurity.com/contrast-community-edition

[8] DAST

Dynamic Application Security Testing is a “black-box” security testing method that finds security issues in the applications by seeing how the application responds to specially crafted requests that mimic attacks.

During our engagement the following DAST tools were used:

- Burp Pro Scan[10]
- OWASP ZAP Scan[11]
- Nikto[12]
- Wapiti[13]

[8.1] OWASP Zed Attack Proxy (ZAP)

OWASP ZAP is an open-source web application security scanner. It is one of the most active Open Web Application Security Project (OWASP) projects and has been given Flagship status.

ZAP Scanning Report

Summary of Alerts **Generated on Fri, 2 Jul 2021 03:50:08**

Risk Level	Number of Alerts
High	0
Medium	1
Low	2
Informational	4

Alerts

Name	Risk Level	Number of Instances
X-Frame-Options Header Not Set	Medium	14
Cross-Domain JavaScript Source File Inclusion	Low	36
X-Content-Type-Options Header Missing	Low	37
Content-Type Header Missing	Informational	3
Information Disclosure - Suspicious Comments	Informational	44
Timestamp Disclosure - Unix	Informational	70

Above is the relevant screenshot from the ZAP Scanning Report. Full report is enclosed with this document.

10 https://portswigger.net/burp/pro
11 https://www.zaproxy.org
12 https://github.com/sullo/nikto
13 https://github.com/wapiti-scanner/wapiti

[8.2] Burp Scan

Burp Suite is an integrated platform for performing security testing of web applications. Its various tools work seamlessly together to support the entire testing process, from initial mapping and analysis of an application's attack surface, through to finding and exploiting security vulnerabilities.

Burp Scanner Report

Burp Suite Professional

Summary

The table below shows the numbers of issues identified in different categories. Issues are classified according to severity as High, Medium, Low or Information. This reflects the likely impact of each issue for a typical organization. Issues are also classified according to confidence as Certain, Firm or Tentative. This reflects the inherent reliability of the technique that was used to identify the issue.

		Confidence			
		Certain	Firm	Tentative	Total
Severity	High	0	0	0	0
	Medium	0	0	0	0
	Low	0	0	4	4
	Information	158	2	0	160

The chart below shows the aggregated numbers of issues identified in each category. Solid colored bars represent issues with a confidence level of Certain, and the bars fade as the confidence level falls.

Contents

1. Vulnerable JavaScript dependency

1.1. http://127.0.0.1:8080/babylon/sound-publish.html
1.2. http://127.0.0.1:8080/babylon/sound-subscribe.html
1.3. http://127.0.0.1:8080/babylon/sound-test.html
1.4. http://127.0.0.1:8080/babylon/video-test.html

Above is the relevant screenshot from the Burp Scanner Report. Full report is enclosed with this document.

[8.3] Nikto

Nikto is an Open Source (GPL) web server scanner which performs comprehensive tests against web servers for multiple items.

Below is the output of nikto scan.

```
- Nikto v2.1.5
---------------------------------------------------------------------------
+ Target IP:          127.0.0.1
+ Target Hostname:    localhost
+ Target Port:        8080
+ Start Time:         2021-07-01 20:53:07 (GMT-4)
---------------------------------------------------------------------------
+ Server: No banner retrieved
+ The anti-clickjacking X-Frame-Options header is not present.
+ Uncommon header 'content-disposition' found, with contents: inline;filename=f.txt
+ No CGI Directories found (use '-C all' to force check all possible dirs)
+ Allowed HTTP Methods: OPTIONS, GET, HEAD, POST
+ 6544 items checked: 0 error(s) and 3 item(s) reported on remote host
+ End Time:           2021-07-01 20:53:37 (GMT-4) (30 seconds)
---------------------------------------------------------------------------
+ 1 host(s) tested
```

[8.4] Wapiti

Wapiti is a web application vulnerability scanner that audits the security of web applications or websites. It performs "black-box" scans (it does not study the source code) of the web application by crawling the webpages of the deployed webapp, looking for scripts and forms where it can inject data. Once it gets the list of URLs, forms and their inputs, Wapiti acts like a fuzzer, injecting payloads to see if a script is vulnerable.

```
    __  __              _ _   _ _____
   / / /\ \ \__ _ _ __ (_) |_(_)___ /
   \ \/  \/ / _` | '_ \| | __| | |_ \
    \  /\  / (_| | |_) | | |_| |___) |
     \/  \/ \__,_| .__/|_|\__|_|____/
                  |_|
Wapiti-3.0.1 (wapiti.sourceforge.net)
[*] Saving scan state, please wait...

 Note
========
This scan has been saved in the file /home/techie/.wapiti/scans/127.0.0.1_8080_folder_09990dc7.db
[*] Wapiti found 37 URLs and forms during the scan
[*] Loading modules:
        mod_crlf, mod_exec, mod_file, mod_sql, mod_xss, mod_backup, mod_htaccess, mod_blindsql, mod_permanentxss,
mod_nikto, mod_delay, mod_buster, mod_shellshock, mod_methods, mod_ssrf

[*] Launching module exec

[*] Launching module file

[*] Launching module sql

[*] Launching module xss

[*] Launching module ssrf

[*] Launching module blindsql

[*] Launching module permanentxss

Report
------
A report has been generated in the file /home/techie/.wapiti/generated_report
Open /home/techie/.wapiti/generated_report/127.0.0.1_8080_07022021_0037.html with a browser to see this report.
```

Wapiti did not find any vulnerabilities.

[9] Manual Code review and custom tooling

Since the standard tooling did not find any vulnerabilities we had to write our own. For that we had to inspect code-base manually in order to try to find coding practices that could lead us to possible misuse of legitimate functions.

Manual code review consisted mainly of inspecting code snippets to check if they contained known patterns that could be exploited. Those included, but were not limited to: ObjectInputStream, Class.forName method, compareTo(), readResolve() java.lang.runtime.Runtime.getRuntime, java.lang.runtime.Runtime.exec java.io.FileInputStream, java.io.FileOutputStream, java.io.FileReader, java.io.FileWriter, readObject(), readValue(), finalize(), invoke(), Object.equals(), Object.hashCode(), compare(), as well as enumerating code snippets that deal with user input.

Additionally, as the user input is mainly dealt through Jackson we extensively checked code to see if there are known Jackson patterns that could lead to deserialization vulnerabilities. We also checked the code containing for us interesting snippets like:

- com.fasterxml.jackson.databind.ObjectMapper
- ObjectMapper mapper = new ObjectMapper();
- objectMapper.enableDefaultTyping();
- @JsonTypeInfo(use=JsonTypeInfo.Id.CLASS, include=JsonTypeInfo.As.PROPERTY, property="@class")
- public Object message;
- mapper.readValue(data, Object.class);

At this point we found no codding patterns that could be used maliciously, so our efforts turned to fuzzing.

[9.1] Custom fuzzers

To limit the scope of fuzzing we first manually checked for possible character escapes in Java, javascript and html as well as unicode, hex code and html entity interpretations within the application.

While injecting unicode character displayed some inconsistent behavior in interpretations of the characters, we deemed none of it exploitable, more so as the usage of unicode html angle brackets was properly sanitized through regex, while html escapes, though displayed in our javasrcipt/html client properly, were not interpreted in it and more importantly within the VRSpace application were treated as a string, meaning that “<” and “<” were displayed “as is”, same with the “>” and “>”, instead of “<” and “>” respectively.

Sanitization was done through clean regex, so our attempts at regexsploit were unsuccessful. Clean regex also meant that whole classes of javascript injection vulnerabilities were mitigated.

Main user interaction with the application is done through json objects delivered through websockets. We mapped all requests and responses through websockets and started writing our own fuzzers.

The first fuzzer was based on the python library called kitty[14] with messages tailored to our application, while the response monitoring was done through ZAP proxy and application's console. This approach did not yield any results as the application held well.

Our next attempt was slightly more precise in that we made a custom client in javascript/html that would serve as a multi-function tool.

Above is the screenshot of our custom tool.

This approach led us to finding minor bugs, that could not be exploited within this implementation.
However, we deemed it prudent to report them, as they could possibly be exploited in later implementations that could be built upon this one.

Examples of this included some parameters of json object ("changes" and "properties" within it) not being thoroughly validated as is shown below:

14 https://github.com/cisco-sas/kitty

[1]
{"object":{"Client":0},"changes":{"properties":{"<img src=1 onerror=alert(1)>":"string" ,"number":123.456}}}

[2]
{"object":{"Client":0},"changes":{"properties":{"string":"string","<img src=1 onerror=alert(1)>" :123.456}}}

[3]
{"object":{"Client":0},"changes":{"properties":{"string":"<img src=1 onerror=alert(1)>","number":123.456}}}

[4]
{"object":{"Client":0},"changes":{"<marquee onscroll=confirm(1)>":"Scroll"}}

[5]
{"object":{"Client":0},"changes":{"<img src=1 onerror=alert(1)>":"<img src=1 onerror=alert(1)>"}}

[6]
{"object":{"Client":0},"changes":{"userHeight":1.8,"<img src=1 onerror=alert(1)>":"<img src=1 onerror=alert(1)>"}}

[7]
{"object":{"Client":0},"changes":{"wrote":"a","<img src=1 onerror=alert(1)>":"<img src=1 onerror=alert(1)>"}}

[8]
{"object":{"Client":0},"changes":{"name":"changed_name", "<img src=1 onerror=alert(1)>":"<img src=1 onerror=alert(1)>"}}

[9]
{"object":{"Client":0},"changes":{"position":{"<img src=1 onerror=alert(1)>":"<img src=1 onerror=alert(1)>","x":-280,"y":0.00800665094814036,"z":-260},"rotation":{"x":0.0013085591195910974,"y":0.8224182792713782,"z":0}}}

[10]
{"object":{"Client":0},"changes":{"position":{"x":-280,"y":0.00800665094814036,"z":-260},"rotation":{"<img src=1 onerror=alert(1)>":"<img src=1 onerror=alert(1)>","x":0.0013085591195910974,"y":0.8224182792713782,"z":0}}}

All the requests above would be forwarded to other clients in the same world and some [1 and 2] would be served to the new clients that are connecting later and entering the same world. However, within this implementation the application would note an error in the browser, “vrspace-ui.js:2474:15”, with the message: “Ignoring unknown event to [object Object]: <img src=1 onerror=alert(1)>”.

The last two examples [9 and 10] would produce no errors in browser, however javascript would not be executed, though all the messages would be forwarded as they were written.

[10] Stress testing

In order to test the number of websocket connections the following tools were used: wsstat, curl and a custom javascript client. All showed similar results.

[10.1] Wsstat

Wsstat[15] is a websocket monitoring and visualization tool.
It aims to make diagnosing problems and understanding websocket infrastructure easy and visual.

Above is the screenshot showing the tool in action. It shows the resource exhaustion of the server

[10.2] Curl.sh

We get similar results with a bash script that is based on the curl with the header specified to upgrade to websocket connection. We get resource exhaustion at around 3960 served connections.

15 https://github.com/Fitblip/wsstat

```bash
#!/bin/bash
mkdir stress
i=0
j=$1
if [$1 == ""] || [$2 == ""]; then
        echo "Usage: ./socket-stress connections timeout"
        echo "  [eg. ./socket-stress 1500 45]"
        echo "  [timeout is in seconds]"
exit
fi
while [ $i -le $j ]
do
echo Connection: $i
  ((i++))
timeout $2 \
curl --include \
     --no-buffer \
     --header "Connection: Upgrade" \
     --header "Upgrade: websocket" \
     --header "Host: localhost:8080" \
     --header "Origin: http://localhost:8080" \
     --header "Sec-WebSocket-Key: SGVsbG8sIHdvcmxkIQ==" \
     --header "Sec-WebSocket-Version: 13" \
     http://localhost:8080/vrspace --output ./stress/response-$i.json  > /dev/null 2>&1 &
done
exit 0
```

```
.0,"y":0.0,"z":0.0},"active":true,"sceneProperties":{"range":2000.0,"resolution":10.0,"size":1000,"timeout":30000}},"timestamp":"2021-07-0
2021-07-02 16:09:40.378  INFO 2769 --- [o-8080-exec-180] org.vrspace.server.core.SessionManager   : New session: d05756ec-f376-34bd-4fb8-1
user null sessions active 3960
2021-07-02 16:09:40.700 ERROR 2769 --- [o-8080-Acceptor] org.apache.tomcat.util.net.Acceptor      : Socket accept failed

java.io.IOException: Too many open files
        at java.base/sun.nio.ch.ServerSocketChannelImpl.accept0(Native Method) ~[na:na]
        at java.base/sun.nio.ch.ServerSocketChannelImpl.accept(ServerSocketChannelImpl.java:533) ~[na:na]
        at java.base/sun.nio.ch.ServerSocketChannelImpl.accept(ServerSocketChannelImpl.java:285) ~[na:na]
        at org.apache.tomcat.util.net.NioEndpoint.serverSocketAccept(NioEndpoint.java:574) ~[tomcat-embed-core-9.0.45.jar!/:na]
        at org.apache.tomcat.util.net.NioEndpoint.serverSocketAccept(NioEndpoint.java:80) ~[tomcat-embed-core-9.0.45.jar!/:na]
        at org.apache.tomcat.util.net.Acceptor.run(Acceptor.java:106) ~[tomcat-embed-core-9.0.45.jar!/:na]
        at java.base/java.lang.Thread.run(Thread.java:829) ~[na:na]

2021-07-02 16:09:42.302 ERROR 2769 --- [o-8080-Acceptor] org.apache.tomcat.util.net.Acceptor      : Socket accept failed

java.io.IOException: Too many open files
        at java.base/sun.nio.ch.ServerSocketChannelImpl.accept0(Native Method) ~[na:na]
        at java.base/sun.nio.ch.ServerSocketChannelImpl.accept(ServerSocketChannelImpl.java:533) ~[na:na]
        at java.base/sun.nio.ch.ServerSocketChannelImpl.accept(ServerSocketChannelImpl.java:285) ~[na:na]
        at org.apache.tomcat.util.net.NioEndpoint.serverSocketAccept(NioEndpoint.java:574) ~[tomcat-embed-core-9.0.45.jar!/:na]
        at org.apache.tomcat.util.net.NioEndpoint.serverSocketAccept(NioEndpoint.java:80) ~[tomcat-embed-core-9.0.45.jar!/:na]
        at org.apache.tomcat.util.net.Acceptor.run(Acceptor.java:106) ~[tomcat-embed-core-9.0.45.jar!/:na]
        at java.base/java.lang.Thread.run(Thread.java:829) ~[na:na]
```

As a side note, socket size test showed websocket message being limited at 8192 bytes in the server-0.2.3-SNAPSHOT.jar version.

[11] Dependency Check

Dependency-Check[16] is a Software Composition Analysis (SCA) tool that attempts to detect publicly disclosed vulnerabilities contained within a project's dependencies. It does this by determining if there is a Common Platform Enumeration (CPE) identifier for a given dependency. If found, it will generate a report linking to the associated CVE entries.

16 https://github.com/jeremylong/DependencyCheck

Below is a screenshot from the report generated by Dependency-Check. Full report is enclosed with this document.

Summary

Display: Showing Vulnerable Dependencies (click to show all)

Dependency	Vulnerability IDs	Package	Highest Severity	CVE Count	Confidence	Evidence Count
server-0.2.3-SNAPSHOT.jar: bcprov-jdk15on-1.64.jar	cpe:2.3:a:bouncycastle:bouncy-castle-crypto-package:1.64:*:*:*:*:*:*:* cpe:2.3:a:bouncycastle:bouncy_castle_crypto_package:1.64:*:*:*:*:*:*:* cpe:2.3:a:bouncycastle:legion-of-the-bouncy-castle:1.64:*:*:*:*:*:*:* cpe:2.3:a:bouncycastle:legion-of-the-bouncy-castle-java-crytography-api:1.64:*:*:*:*:*:*:* cpe:2.3:a:bouncycastle:the_bouncy_castle_crypto_package_for_java:1.64:*:*:*:*:*:*:*	pkg:maven/org.bouncycastle/bcprov-jdk15on@1.64 pkg:maven/org.netbeans.external/bcprov@RELEASE113	MEDIUM	1	Low	61
server-0.2.3-SNAPSHOT.jar: neo4j-java-driver-4.0.2.jar (shaded: io.netty:netty-buffer:4.1.50.Final)	cpe:2.3:a:netty:netty:4.1.50:*:*:*:*:*:*:*	pkg:maven/io.netty/netty-buffer@4.1.50.Final	MEDIUM	3	Highest	9
server-0.2.3-SNAPSHOT.jar: neo4j-java-driver-4.0.2.jar (shaded: io.netty:netty-codec:4.1.50.Final)	cpe:2.3:a:netty:netty:4.1.50:*:*:*:*:*:*:*	pkg:maven/io.netty/netty-codec@4.1.50.Final	MEDIUM	3	Highest	9
server-0.2.3-SNAPSHOT.jar: neo4j-java-driver-4.0.2.jar (shaded: io.netty:netty-common:4.1.50.Final)	cpe:2.3:a:netty:netty:4.1.50:*:*:*:*:*:*:*	pkg:maven/io.netty/netty-common@4.1.50.Final	MEDIUM	3	Highest	9
server-0.2.3-SNAPSHOT.jar: neo4j-java-driver-4.0.2.jar (shaded: io.netty:netty-handler:4.1.50.Final)	cpe:2.3:a:netty:netty:4.1.50:*:*:*:*:*:*:*	pkg:maven/io.netty/netty-handler@4.1.50.Final	MEDIUM	3	Highest	9
server-0.2.3-SNAPSHOT.jar: neo4j-java-driver-4.0.2.jar (shaded: io.netty:netty-resolver:4.1.50.Final)	cpe:2.3:a:netty:netty:4.1.50:*:*:*:*:*:*:*	pkg:maven/io.netty/netty-resolver@4.1.50.Final	MEDIUM	3	Highest	9
server-0.2.3-SNAPSHOT.jar: neo4j-java-driver-4.0.2.jar (shaded: io.netty:netty-transport:4.1.50.Final)	cpe:2.3:a:netty:netty:4.1.50:*:*:*:*:*:*:*	pkg:maven/io.netty/netty-transport@4.1.50.Final	MEDIUM	3	Highest	9
server-0.2.3-SNAPSHOT.jar: spring-core-5.2.14.RELEASE.jar	cpe:2.3:a:pivotal_software:spring_framework:5.2.14:release:*:*:*:*:*:* cpe:2.3:a:springsource:spring_framework:5.2.14:release:*:*:*:*:*:* cpe:2.3:a:vmware:spring_framework:5.2.14:release:*:*:*:*:*:* cpe:2.3:a:vmware:springsource_spring_framework:5.2.14:release:*:*:*:*:*:*	pkg:maven/org.springframework/spring-core@5.2.14.RELEASE	HIGH	1	Highest	31

Furthermore, we inspected currently open bugs in projects integrated within VRSpace application, namely Jackson[17], Tomcat[18] and neo4j[19] and did not find any approach that could compromise the application, more so as the potential vulnerability would not necessarily change the structure of the VRSpace application since the potential vulnerability would be in a third party component.

We also tested all the imported classes from the application for potential Jackson gadget chains, albeit not extensively, during our fuzzing tests, as well as tested the application to check if it uses previously known and blacklisted gadgets[20]. However, multiple prerequisites were not satisfied[21][22] in order for this approach[23] to be feasible.

17 https://github.com/FasterXML/jackson-databind/issues/
18 https://bz.apache.org/bugzilla/buglist.cgi?no_redirect=0&quicksearch=tomcat
19 https://github.com/neo4j/neo4j/issues/
20 https://github.com/FasterXML/jackson-databind/blob/master/src/main/java/com/fasterxml/jackson/databind/jsontype/impl/SubTypeValidator.java
21 https://cowtowncoder.medium.com/on-jackson-cves-dont-panic-here-is-what-you-need-to-know-54cd0d6e8062#da96
22 https://blog.doyensec.com/2019/07/22/jackson-gadgets.html
23 https://adamcaudill.com/2017/10/04/exploiting-jackson-rce-cve-2017-7525/