Fine penetration tests for fine websites

Dr.-Ing. Mario Heiderich, Cure53
Bielefelder Str. 14
D 10709 Berlin
cure53.de · mario@cure53.de

Pentest-Report CNCF Jaeger 04.-05.2019

Cure53, Dr.-Ing. M. Heiderich, M. Wege, MSc. N. Krein, MSc. D. Weißer, J. Larsson, BSc. J. Hector

Index

Dr.-Ing. Mario Heiderich, Cure53
Bielefelder Str. 14
D 10709 Berlin
cure53.de · mario@cure53.de

Introduction

"As on-the-ground microservice practitioners are quickly realizing, the majority of operational problems that arise when moving to a distributed architecture are ultimately grounded in two areas: networking and observability. It is simply an orders of magnitude larger problem to network and debug a set of intertwined distributed services versus a single monolithic application."

From https://www.jaegertracing.io/

This report documents the findings of a security assessment targeting the CNCF Jaeger software, a distributed Tracing System released as open source by Uber. Carried out by Cure53 in spring 2019, this project entailed a source code audit and a penetration test.

As for the resources, six members of the Cure53 took part in the project. The allocated time budget was eighteen days and these were all dedicated to project tasks and objectives in late April and early May of 2019. The targeted release - equating to the scope of this examination - was CNCF Jaeger 1.11.0. It can be noted that the source code was taken from the publicly available Github repository, as the project has an open source character. Consequently, the chosen methodology was white-box and paired a comprehensive code audit with a classic penetration testing against a system put forward by the maintainers of the scoped Jaeger version.

Besides the actual software, several of the available clients were in scope as well. Note that given the time available to the testers, not all clients could be audited. In addition, the Kubernetes-related code and configuration files were also inspected. All in all, Cure53 did not manage to find any notable threats. Only a very small array of three general weaknesses with negligible risk potential could be distinguished. Nevertheless, even though the threat model for CNCF Jaeger is rather "generous" and not many items could be pointed out as "risks", experience has shown that attackers, once a system like CNCF Jaeger is exposed to the public Internet, benefit from missing security controls. Therefore, Cure53 found it crucial to also focus on hardening strategies that can make the software even better and safer.

The next section in this report will elaborate on the scope details, list the release version and repositories in scope. Next, the report will detail the chosen methodology and describes various testing approaches, tasks and strategies to illuminate the span and scope of the coverage reached by this Cure53 assessment. This means that a checklist of the analyzed items is included to document the baseline of this evaluation of the CNCF Jaeger security posture. The ensuing section is linked with the above statements about the hardening needs, as the report furnishes on general advice and hardening recommendation across the areas possibly prone to attacks. A detailed discussion of the

Dr.-Ing. Mario Heiderich, Cure53
Bielefelder Str. 14
D 10709 Berlin
cure53.de · mario@cure53.de

aforementioned three findings follows. Last but not least, the report closes with a conclusion, in which Cure53 summarizes their overall impressions about the CNCF Jaeger's general security posture.

Scope

- **CNCF Jaeger**
  - In scope was the core system for CNCF Jaeger (1.11.0)
    - https://github.com/jaegertracing/jaeger/releases
  - Also investigated were the relevant client libraries (a selection done by Cure53)
    - https://github.com/jaegertracing/jaeger-client-python
    - https://github.com/jaegertracing/jaeger-client-node
    - https://github.com/jaegertracing/jaeger-client-go
    - https://github.com/jaegertracing/jaeger-client-cpp
  - Kubernetes-related code and config
    - https://github.com/jaegertracing/jaeger-kubernetes
  - Cure53 was given access to a dedicated setup for the purpose of completing the penetration test

Test Methodology

The following paragraphs describe the testing methodology used during the audit of the Jaeger codebase and its related client-libraries. The test was divided into two phases, each fulfilling different goals. In the first phase, the focus was on manual source code reviews needed to spot insecure code patterns. In this realm, issues around memory corruption, information leakage or similar flaws can be found. During the second phase, it was evaluated whether the stated security goals and premise can, in fact, withstand real-life attack scenarios.

Part 1. Manual Code Auditing

This section lists the steps that were undertaken during the first phase of the audit against the Jager software compound. It describes the key aspects of the manual code audit. Since no major issues were spotted, the list portraits the thoroughness of the penetration test and attests a high quality of the project.

- The Jaeger codebase was checked for potentially vulnerable sinks in the locations were user-input is parsed and handled.
- Client-libraries were audited to check how a connection is established to either the agent daemon or the collector. The polling of the sampling strategy and submitting of trace data were examined.

Dr.-Ing. Mario Heiderich, Cure53
Bielefelder Str. 14
D 10709 Berlin
cure53.de · mario@cure53.de

- Additionally, the code for creating spans and tags was audited to ensure that the data is stored in a safe manner. Flaws in this area could otherwise lead to injection-based attacks.
- Client libraries were also checked for OS interactions (e.g. logging or binary execution), however, the logging was either to null or console. No further OS interactions were found.
- Special attention was given to the C++ client library (due to the nature of the C++ language). Dynamic memory allocations, unsafe use of typical C APIs and possible integer overflows were checked. However, the strict use of modern C++ constructs prevents common pitfalls on the CNCF Jaeger scope.
- HTTP handler functions of the collector and agent daemon were audited for logical flaws and general handling of untrusted input. Additionally, it was checked whether proper ACL checks were in place. More information on this subject can be found in JT-01-001.

Part 2. Code-Assisted Penetration Testing

The following list documents the distinguishable steps taken during the second part of the test. A code-assisted penetration test was executed against the pre-configured Jaeger cluster provided by the development team. Since only a few miscellaneous issues were found during the first part of the audit, this additional approach was used to ensure maximum coverage of the originally defined attack surface.

- The admin web interface - which allows to view and submit traces - was tested for common, web-related server and client issues like CSRF, XSS and injections. Besides a minor DoS issue, no problems with the handling of user-input have been found.
- The accepted *Content-Types* were evaluated for possibly exploitable avenues. The *gzip Content-Type* was examined in relation to the so-called *gzip*-bombs but no such weaknesses could be identified.
- The JSON *upload* feature was examined for XSS vulnerabilities with the use of maliciously formatted JSON files. It has become apparent that all content is being escaped properly and therefore no such weaknesses seem to exist.
- The default set of tests for discovering problems in the web security area was applied. This included general fuzzing and attempts at a targeted injection of content to appear in log-files or OS-level system calls. No such problems were found.

Dr.-Ing. Mario Heiderich, Cure53
Bielefelder Str. 14
D 10709 Berlin
cure53.de · mario@cure53.de

Focus Areas and General Recommendations

The following section talks about the focus areas previously discussed by Cure53 with the maintainers and overall outlined by the development team. Recommendations for further improving the security of the software are given due to the already noted generous nature of the threat model. In that sense, Cure53 comments on the fact that responsibility for security measures is allocated to other stakeholders in the stack and argues that this might have certain security-implications in various realms. This is paired with an emphasis on hardening advice.

Input Manipulation and Code Injection Attacks

During the audit of Jaeger, considerable attention centered on common security issues that are usually related to server-side software. In essence, any functionality that processes user-input was analyzed and checked for consequences of malicious input. The testing methodology entailed extensive code assisted manual input manipulation tests reliant on functionality exposed by the web UI or reachable with the Jaeger clients in scope.

It was quickly noticed that the Jaeger codebase clearly avoids common programming mistakes, such as having the user-controlled arguments that are passed to calls through *os.exec* or *ioutils.ReadFile.* These easily result in command execution of file disclosure if done incorrectly and here were eliminated successfully.

The general programming style was found to be very defensive. Error codes are checked thoroughly, parsing (for example for JSON *Content-Types)* is done with care and bailouts are done early. Nearly all models implement extensive unit-tests that cover multiple forms of unconventional input and make sure that the defined methods work as expected. This extends to unmarshaling of common data-types but also thoroughly tests the unpacking of specially implemented formats such as *trace-IDs* and similar. Generally, Jaegers codebase does not leave much room for errors, which is reflected in the low number of vulnerabilities found by Cure53.

Concerning the given client-side software, Cure53 conducted audits of the client-libraries as well. They use a straightforward and simple interface to communicate either with the local agent or the collector itself. The communication consists of reporting spans and polling the sampling strategy provided by the collector. Given the simple design, not much room for errors is left. Furthermore, the client-libraries provide an API to associate tags with span data; these were audited to ensure that malicious input-data does not allow for an attack corrupting the span data. The client-libraries use the proper structures, provided by the implementation language, to store key and value pairs.

Dr.-Ing. Mario Heiderich, Cure53
Bielefelder Str. 14
D 10709 Berlin
cure53.de · mario@cure53.de

Moreover, the code is well-written and follows a clean coding style which further reduces common pitfalls.

Logic Bugs, ACL & Integrity Tests, Privilege Escalation

One major aspect specified by the development team as a priority was the verification of access controls. This warranted checking their integrity and locating logic bugs that would allow for escalating privileges.

Surprisingly, it was found that there are no access controls in the analyzed source code. While there are access roles defined in the Kubernetes configuration, these were not found to be applied anywhere. Similarly, there were no message integrity checks in the implementation. Since there was no application of such tests, in turn, no logic bugs bypassing the probes could be identified.

The exposed service hosting the Jaeger API was analyzed for potential privilege escalation issues that could reside within the service itself, as well as in the hosting infrastructure. However, no privilege escalation issues were discovered: neither in the realm of the service nor through pod-to-pod execution. At the same time, the current configuration raised some concerns for the testers due to the adopted network topology. The Kubernetes cluster has no network policy object configured and relies on a very lax network separation design. The defense-in-depth ticket JT-01-003 was created to shed light on these aspects.

The further possible shortcoming was encountered while investigating the authentication scheme adopted by Jaeger. As of now, the only way to support authentication is to rely on third-party services such as *Oauth*-proxies or native Kubernetes authentication. The lack of a natively supported authentication mechanism is considered a bad security practice and should be rectified by the Jaeger maintainers. JT-01-001 was created to address this.

Security aspects of the general Kubernetes deployment

During this audit, the overall security aspects of the current Kubernetes orchestration were analyzed. The focus was placed on discovering weak configurations that could be leveraged by an attacker to gain unwanted access to the deployment pipelines, the Kubernetes cluster, as well as to running pods and services.

The images used throughout the configuration are handled in a safe manner: all images are downloaded from reputable sources and have the appropriate signatures in place so that the authenticity of the downloaded image is always validated.

Dr.-Ing. Mario Heiderich, Cure53
Bielefelder Str. 14
D 10709 Berlin
cure53.de · mario@cure53.de

The Kubernetes cluster configuration was analyzed as regards known security issues that reside in the Kubernetes master and corresponding nodes. The configuration used by Jaeger is more or less a standard installation of Kubernetes. The configured namespaces and the attached pods were checked for common pod-to-pod execution issues as well as pod-to-master vectors, which could be leveraged by an attacker who has gained an initial foothold in either a service or a pod inside of the cluster. No immediate weak configuration issues were discovered apart from the miscellaneous ticket described in JT-01-003.

Programmatic DoS Attacks, Memory Exhaustion

Another relevant part of the audit against Jaeger revolved around potential sinks that could cause the Jaeger setup to crash and become unrecoverable. Generally, this was also covered by further input manipulation tests, especially on areas that can allow submission of large values for input variables, files and compressed data. However, it was quickly found that, for example for *gzip*-compression, standard libraries were in place and prevented such issues on their own already.

Still, the testers managed to find a minor DoS issue concerning the bindings for the *Elasticsearch* service, as described in JT-01-002. Although Jaeger uses a robust architecture that takes care of all services to be restarted consequently, this was still treated as an input-validation issue that was worth reporting. Even with the focus on more configurational issues, the auditors were not able to uncover methods to deterministically cause a Denial-of-Service in the provided Jaeger setup.

Web-User Interface Security

Since Jaeger offers an administrative web interface, this item had to be analyzed from a security perspective. Although it is deployed without any form of authentication where the end user is expected to sufficiently shield it from outside attackers, malicious vectors may still exist. This concerns reflected XSS, code injection issues and other common vulnerabilities that are typically present in web applications. As such, the testers conducted extensive fuzzing and input-manipulation tests to spot such issues, for example in the rendered templates and other areas that receive user-input, such as the REST API.

Especially the trace data import via JSON files received considerable attention as it allows arbitrarily chosen values for each field. When displaying the data, it was found that the UI encodes all characters correctly in their HTML entities, which leaves no room for further exploitation. As the UI is kept rather simple and does not implement any other complex logic, not much other attack surface existed. The auditors further looked for

Dr.-Ing. Mario Heiderich, Cure53
Bielefelder Str. 14
D 10709 Berlin
cure53.de · mario@cure53.de

places where user-input could end up in files residing in the file-systems and checked how these could interfere with other services or applications.

Another integral part of the UI security penetration test was the REST API which is used to fetch data. Except for the minor issues around the absence of rate-limiting and the leakage of internal IPs in verbose error messages, the testers could not determine any serious web security issues.

Service Robustness & Proneness to Crashes

During the audit, the testers assessed the overall robustness and scalability of all components attached to the Jaeger scope, in order to discover potential issues that could possibly lead to unexpected crashes or similar unwanted behavior.

In order to achieve good coverage, all services were analyzed by looking for common misconfigurations that could eventually lead to Denial-of-Service issues or over-allocation of resources within the Kubernetes cluster. No configuration issues were discovered to affect the robustness and scalability of the orchestration used by Jaeger. The discovered DoS (filed as JT-01-002) did not affect the overall performance of the cluster and triggered no scaling issues that would render the entire cluster unstable.

Miscellaneous Issues

This section covers those noteworthy findings that did not lead to an exploit but might aid an attacker in achieving their malicious goals in the future. Most of these results are vulnerable code snippets that did not provide an easy way to be called. Conclusively, while the vulnerability is present, an exploit might not always be possible.

JT-01-001 Collector: Missing authentication on data submission *(Info)*

A central component of the Jaeger architecture is the collector. It receives the trace reports from all *agents/micro* services and stores the data into a data store. The collector exposes multiple HTTP endpoints, for example */api/traces* or */api/v1/spans*. The endpoints are used directly by the agent daemon or micro-services to submit trace data. These endpoints, however, do not perform any kind of authentication. Thus, a micro-service that is vulnerable to a sophisticated Server-Side Request Forgery (SSRF)[1] attack, can be abused to submit malicious trace data to the collector.

Although the developers encourage the end-users to take the appropriate steps based on their deployment, it is still recommended to implement some form of authentication. For example, token-based access, to restrict collector access to only authenticated

[1] https://www.owasp.org/index.php/Server_Side_Request_Forgery

Dr.-Ing. Mario Heiderich, Cure53
Bielefelder Str. 14
D 10709 Berlin
cure53.de · mario@cure53.de

agents/micro-services, could be added. A default authentication can also be implemented as an opt-out feature, which gives users the freedom to go with a different route.

JT-01-002 UI: DoS on *Elasticsearch* due to large input values *(Info)*

The UI relies on an *Elasticsearch* instance to query for trace data. A user can supply a *loopback* or *start* and *end* params, which will determine which time span the search should cover. It was found that by supplying large values for the *loopback* param or by creating a big timespan, one can cause the underlying *elasticsearch* crashes. Illustrative requests can be seen below.

Example Request 1:

```
GET /api/traces?start=1&end=1556624710030000&operation=multiRead&service=jaeger-
query HTTP/1.1
Host: 139.178.82.82:31112
Connection: close
```

Response:

```
HTTP/1.1 500 Internal Server Error
Content-Type: text/plain; charset=utf-8
Content-Length: 450726

{"data":null,"total":0,"limit":0,"offset":0,"errors":[{"code":500,"msg":"Search
service failed: Post http://elasticsearch:9200/jaeger-span-2019-04-30%2Cjaeger-
span-2019-04-29[...]%2Cjaeger-span-1970-01-01/span/_search?
ignore_unavailable=true: net/http: HTTP/1.x transport connection broken: write
tcp 10.233.70.27:49502-\u003e10.233.65.22:9200: write: connection reset by
peer"}]}
```

Example Request 2:

```
GET /api/dependencies?endTs=1&lookback=9218999999999 HTTP/1.1
Host: 139.178.82.82:31112
```

Response:

```
HTTP/1.1 500 Internal Server Error
Content-Type: text/plain; charset=utf-8
Content-Length: 3521520

{"data":null,"total":0,"limit":0,"offset":0,"errors":[{"code":500,"msg":"Failed
to search for dependencies: Post http://elasticsearch:9200/jaeger-dependencies-
1970-01-01%2Cjaeger-dependencies-1969-12-31%2C[...]%2Cjaeger-dependencies-1677-
11-10/dependencies/_search?ignore_unavailable=true: net/http: HTTP/1.x transport
connection broken: write tcp 10.233.70.27:33606-\u003e10.233.65.22:9200: write:
connection reset by peer"}]}
```

Dr.-Ing. Mario Heiderich, Cure53
Bielefelder Str. 14
D 10709 Berlin
cure53.de · mario@cure53.de

Sample DoS Response:

```
HTTP/1.1 500 Internal Server Error
Content-Type: text/plain; charset=utf-8
Content-Length: 157
Connection: close

{"data":null,"total":0,"limit":0,"offset":0,"errors":[{"code":500,"msg":"Search
service failed: no available connection: no Elasticsearch node available"}]}
```

As seen above, subsequent requests will be denied until the *Elasticsearch* instance is up again. Due to its integration into *K8s*, this period of downtime, however, is only a matter of seconds. Despite not having a direct security impact, the Jaeger team deemed this find useful. It is recommended to add further sanitization methods to ensure that all supplied values are sufficiently bound-checked.

JT-01-003 K8s: Missing defense-in-depth procedures *(Info)*

The analysis of the Kubernetes cluster showed that the orchestration topology lacks defense-in-depth concepts. The current orchestration has a traditional inside vs. outside defense boundary defined. Running this topology design is not considered a sound security practice. If an attacker was to gain a foothold into the Kubernetes cluster through a compromised pod, there is no segmentation in place to limit *ingress/egress* traffic. The analyzed orchestration has multiple namespaces configured but there is no limiting configuration in place to aid network or services segmentation.

As an example, the *Elasticsearch* pod that runs in the namespace *storage* is able to reach all internal pods defined within the cluster, as well as reaching the public Internet. If this pod was to be compromised, it could be used as a pivot point for further attacks throughout the cluster and infrastructure.

Excerpt from the *elasticsearch-0* pod:
An internal resource using the "*example*" namespace is reachable from the "*storage*" namespace.

***Shell* excerpt:**

```
$ curl -L 10.233.29.206:8080
Hello from Vert.x!
```

External resources are available from within the *Elasticsearch* pod.

***Shell* excerpt:**

```
$ curl -L https://cure53.de
<!doctype html><!--
```

Dr.-Ing. Mario Heiderich, Cure53
Bielefelder Str. 14
D 10709 Berlin
cure53.de · mario@cure53.de

```
-->
<html lang="en-US">
    <head>
        <script src="/all.js"></script>[...]
```

It is recommended to inform and educate end-users on how to harden and secure their respective infrastructure when Jaeger is deployed into a running Kubernetes cluster. In order to properly protect all components used by Jaeger, *ingress/egress* traffic configurations should be implemented by default or as a network policy that the end-user can select to either apply or opt out from.

Conclusions

In light of the findings stemming from this 2019 assessment of the Jaeger Tracing system, Cure53 has gained a mixed impression of the examined scope. To give some details, the Cure53 investigation of the Jaeger Tracing system was generously financed by The Linux Foundation / Cloud Native Computing Foundation, which enabled a team consisting of six Cure53 testers to investigate the software system over the course of eighteen days in April and May of 2019. Though a good coverage of almost all components and areas has been achieved, Cure53 was unable to pinpoint any real vulnerabilities in the codebase. At the same time, the general approach to security displayed by the development team has been evaluated as somewhat lacking in the Cure53 team's expert opinion.

On one hand, the general indicators analyzed during the project are very good. In particular, no actual security threats have been identified and only a handful of miscellaneous issues could be spotted. This can be attributed to the high code quality and a well-chosen implementation language, as well as libraries. In addition, positive outcomes can be linked to the characteristics of the deployment and execution environment. On the other hand, the auditors are somewhat concerned about the apparent dismissal of all security mechanisms within the implementation itself. In that sense, the Jaeger project appears nearly void of actual security measures. Everywhere in the codebase and in terms of key properties, a correct and complete configuration of the deployment and execution environment is a precondition and main approach. Such a complete reliance on perimeter-security calls the generally accepted industry practice of defense-in-depth into question. Moreover, it does not signify a capacity to cover possible misconfigurations and the yet unknown, emerging risks.

As already acknowledged above, many positive conclusions can be drawn about the Jaeger Tracing project. Especially the proper design and the clean implementation of the concepts required for a system addressing this particular problem area must be praised. While it was possible to produce a temporary Denial-of-Service state, all attempts at a

Dr.-Ing. Mario Heiderich, Cure53
Bielefelder Str. 14
D 10709 Berlin
cure53.de · mario@cure53.de

complete shutdown of the system were futile. It is theoretically possible to forge trace data, as well as exfiltrate data from any system within a cluster that an attacker may have accessed in some way or another. However, the default configuration makes any further success of the adversaries highly unlikely and quite unfeasible.

The auditors hope that the recommendation of implementing additional defense-in-depth features will be taken to heart. This would make the entire Jaeger system more resilient and can serve as an inspiration for the continued development of the software compound. The testers strongly believe that the well-accepted industry-wide concept that entails moving away from solely relying on perimeter security will be beneficial for the project. As such, it will help achieve a secure system, making Jaeger an even better product. Taking into consideration various non-default deployment scenarios can shift responsibility for the security traits of an installation from the end-users. In any way, profound expertise of the end-users should not be an assumption made lightly by the development team. If no additions to the system are made, at least the documentation should be adjusted to warn the users about the possible, harmful side-effects.

Cure53 would like to thank Gary Brown, Juraci Paixao Kroehling, Kevin Earls, Prithvi Raj and Yuri Shkuro from the CNCF Jaeger team as well as Chris Aniszczyk of The Linux Foundation, for their excellent project coordination, support and assistance, both before and during this assignment. Special gratitude needs to be extended to The Linux Foundation for sponsoring this project.