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

Pentest-Report Linkerd2 & Linkerd2 Proxy 06.2019

Cure53, Dr.-Ing. M. Heiderich, M. Wege, Prof. N. Kobeissi, N. Hippert, J. Larsson,
BSc. J. Hector, BSc. T.-C. "Filedescriptor" Hong

Index

Introduction

"Linkerd is a service mesh for Kubernetes and other frameworks. It makes running services easier and safer by giving you runtime debugging, observability, reliability, and security—all without requiring any changes to your code."

From https://linkerd.io/2/overview/

This report documents the results of a security assessment targeting the Linkerd complex. Carried out by Cure53 in June 2019, this project entailed both a penetration test and a source code audit, which specifically investigated Linkerd, the Linkerd Proxy and the gRPC API bindings. It should also be noted that this security-centered examination was requested and sponsored by The Linux Foundation/CNCF.

As for the resources, a total of seven Cure53 testers completed this project by spending a total of eighteen days on analyzing the scope with a wide range of approaches. The guiding methodology was, by default, white-box because all sources are available publicly. Adhering to the well-established standards, Cure53 followed a specific, two-pronged approach of relying on source code auditing on the one hand, and executing a penetration test, on the other. The latter targeted several remote and local instances where Linkerd is being used in the same way that it would likely be used in production.

To support and facilitate the project, the Linkerd in-house team helped Cure53 with setting up a cluster in which Linkerd was used as intended by the maintainers. This handling was very helpful in getting access to (and awareness of) a realistic test-target. The communications during the test were done in a shared Slack channel, set up by Cure53 and joined by the Linkerd developers from Buoyant. In addition to that, a scope document was shared via Google Drive and then used throughout the test in a collaborative manner. The Linkerd team could add information about areas of interest “on the go”, which resulted in a better familiarity and clarity about scope details and other matters. All these actions contributed to the Cure53’s capacity of completing all objectives on time and to a high standard. With good communications in place, Cure53 was able to reach good coverage over the security relevant areas.

It became clear to Cure53 quite fast that the Linkerd codebase and implementation are very robust from a security standpoint. The tested item made a very good impression and very few, rather marginal issues were spotted. Nevertheless, the two spotted flaws relate to general weaknesses in the web interface, which could be hardened a bit better in Cure53’s expert opinion.

This report will now describe the scope in more detail and then moves on to a comprehensive overview of the test methodology and coverage. As not many findings have been spotted, Cure53 uses this section to help the Linkerd maintainer team navigate through the approaches and types of tests completed on the scope. Next up, the two findings will be documented, alongside with recommendations about addressing them in the best way possible. The report then closes with the usual conclusion in which Cure53 describes the test as such, as well as reiterates the results and arrives at a final, broader verdict of this 2019 engagement Conclusions pertinent to the general security posture of the Linkerd and Linkerd Proxy projects ensue.

Scope

- **Linkerd & Linkerd Proxy**
  - https://github.com/linkerd/linkerd2
  - https://github.com/linkerd/linkerd2-proxy
  - https://github.com/linkerd/linkerd2-proxy-api
- **Documentation**
  - https://linkerd.io/2/reference/
- **A Scope Document was shared between Linkerd Team and Cure53**

Test Methodology

The following paragraphs describe the testing methodology used during the audit of the Linkerd2 and the related, Linkerd2 Proxy-codebase. 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. Usually issues around race conditions, information leakage or similar flaws can be found in this context. 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 Linkerd software compound. It describes the key aspects of the manual code audit. Since no major issues were spotted, the list portrays the thoroughness of the penetration test and attests the impressively high quality of the project.

- The Linkerd documentation was extensively studied to obtain a solid overview of the software compound, in particular its architecture. Cure53 sought to get a grasp on the possibly problematic areas and potential attack surfaces.
- A general audit of the Go codebase for dangerous sinks (such as command execution functions) with unsanitized input was conducted.
- The HTTP handlers of the dashboard web interface were audited for incorrect handling of user-input.
- A more in-depth look was taken at the *tap* interface and how the interaction between Linkerd and the Proxy takes place.
- The file handling code serving static content was checked for common problems but found to be properly dealing with the subject.
- The gRPC API client and gRPC request generation were analyzed for common problems around parameters containing special characters.
- The JSON error handling implementation was audited for typical weaknesses but is avoiding the general traps.

- General route mappings, the *admin* server interfaces/mappings and the Grafana route mappings were extensively checked for edge cases.
- The configuration file download code was evaluated for correct format handling and appears to be doing an impeccable job.
- The control plane APIs were enumerated, tested for information leakage and were confirmed to be read-only across the board.
- The general HTTP handling code was analysed for known shortcomings but none could be pinpointed.
- The HTTP header handling in the Proxy code was verified to function properly and without falling short across errors.
- The React code of the dashboard was checked for perilous patterns - like *dangerouslySetInnerhtml* - that could eventually lead to XSS, along with other typical client-side issues (e.g. client-side path traversal). No controllable parameters could be identified.
- Proxy code for handling inbound and outbound connections has been audited for logical flaws. Special attention was given to header manipulation, which could lead to unintentional routing, as well as to the protocol detection mechanism.
- The Grafana Proxy logic, together with the API handling logic, were investigated for allowing path traversal as well as Host header injections.
- The TLS component was audited for correct handling of secure connections, in particular the parsing and serialising of certificates were scrutinized.
- The *Identity* component was checked for correct handling of X.509 certificates, especially as regards the creation of certificates.
- The *Healthcheck* component was checked for general weaknesses of its implementation concerning maintenance tests and sanity checks.
- The *Transport* component of the Proxy code was audited for its application of TLS towards the handling of network connections.
- The *Identity* component of the Proxy code was checked for correct handling of certificates, especially the parsing and serializing of certificates.
- The DNS component of the Proxy code was checked for logic weaknesses in the relevant aspects resolving domain names.

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 Kubernetes cluster running Linkerd and demo applications 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.

- Initially the test infrastructure was examined and tested for proper connectivity as well as for actually available functionality.
- Open issues and previously found bugs were tracked through the Linkerd repositories, looking for similar errors being present in the tested version.
- The Kubernetes configuration and integration aspects were analyzed, a search for common misconfiguration issues pertaining to Linkerd was performed.
- The running services and their individual configurations, in particular the interfaces, were checked for proper compartmentalization. It was discovered that Linkerd is located in its own namespace but has cluster-wide access.
- A customized fuzzing map was created to verify the general robustness of the services. General as well as specific payloads were created, based on the previously observed Linkerd requests. The payloads yielded a noticeable increase in latency but did not have implications on availability or security.
- *Linkerd-cli* was analyzed to determine its scope of functionality. Cure53 attempted to use it as a source of compromise. Since a corresponding *kubectl.yam*l file is used to connect to the cluster and subsequently install its own control plane API, dependencies and *tap* interfaces, this approach was quickly abandoned.
- *Linkerd-web* was audited for flaws by manipulating requests for certain applications/namespaces. This was done through the addition of malformed headers.
- *Linkerd-proxy-injecto*r and *proxy-auto-inject* were tested by trying to inject bogus and slightly malformed requests into pre-routing.
- *Linkerd-tap* was probed by triggering logical flaws via targeted header manipulation and attempted request breakouts.
- The dashboard web interface has been tested in a black-box fashion by manipulating parameter values to contain special characters. Cure53 wanted to see if this approach can cause any unintended behaviors.
- Header manipulation has been tested by altering existing headers and adding headers in requests to the demo application. It was hoped that unintentional routing can be triggered.

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 a vulnerability is present, an exploit might not always be possible.

LNK-01-001 Dashboard: General HTTP security headers missing *(Info)*

It was found that the Linkerd dashboard is missing certain HTTP security headers in HTTP responses. This does not directly lead to a security issue, yet it might aid attackers in their efforts to exploit other problems. The following list enumerates the headers that need to be reviewed to prevent flaws that can be attributed to mishandled or misconfigured headers.

- **X-Frame-Options**: This header specifies whether the web page is allowed to be framed. Although this header is known to prevent Clickjacking attacks, there are many other attacks which can be achieved when a web page is frameable[1]. It is recommended to set the value to either **SAMEORIGIN** or **DENY**.
- **X-Content-Type-Options**: This header determines whether the browser should perform MIME Sniffing on the resource. The most common attack abusing the lack of this header is tricking the browser to render a resource as an HTML document, effectively leading to Cross-Site-Scripting (XSS).
- **X-XSS-Protection**: This header specifies if the browser's built-in XSS auditors should be activated (enabled by default). Not only does setting this header prevent Reflected XSS, but also helps to avoid the attacks abusing the issues on the XSS auditor itself with false-positives, e.g. Universal XSS[2] and similar. It is recommended to set the value to either **0** or **1; mode=block**.

Overall, missing security headers is a bad practice that should be avoided. It is recommended to add the following headers to every server response, including error responses like *4xx* items.

More broadly, it is recommended to reiterate the importance of having all HTTP headers set at a specific, shared and central place rather than setting them randomly. This should either be handled by a load balancing server or a similar infrastructure. If the latter is not possible, mitigation can be achieved by using the web server configuration and a matching module.

[1] https://cure53.de/xfo-clickjacking.pdf
[2] http://www.slideshare.net/masatokinugawa/xxn-en

LNK-01-002 Dashboard: Missing DNS Rebinding attack protection *(Low)*

It was found that the Linkerd dashboard service, as well as the Grafana dashboard, are vulnerable to a DNS Rebinding attack. This allows an attacker to have a victim visiting an attacker-controlled website. The attacker will then be able to perform requests and retrieve responses on the victim's behalf.

The attack exploits the fact that the dashboard does not restrict the *Host* header for incoming requests. By setting up a script on a domain with a very low TTL value and then quickly rebinding its IP address of localhost, the script will then "bypass" the restrictions normally set by SOP on the browser.

Steps to Reproduce:

- Run Linkerd dashboard service with port 3000 (`linkerd dashboard --port 3000`)
- Go to http://rebind.it:8080/manager.html and change the target port to 3000.
- Click on *"Start Attack"* and wait for a minute.
- The response from the dashboard will be returned, indicating success of a DNS Rebinding attack

It is recommended to provide an option to fixate the dashboard domain. Grafana, for example, provides an *enforce_domain* option to ensure that redirection to the correct domain[3] happens consistently.

[3] https://grafana.com/docs/v3.1/installation/configuration/#enforce_domain

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

Conclusions

Judging by the lack of discovered relevant vulnerabilities and only a few miscellaneous issues, Cure53 has gained a rarely observed and very good impression of the examined Linkerd software complex and its surroundings. This June 2019 Cure53 project clearly demonstrates that the Linkerd product is fully capable of preventing major attacks and should be considered strong against the majority of malicious attempts at a compromise.

To give some context, this security investigation of the Linkerd system was generously funded by The Linux Foundation / Cloud Native Computing Foundation, which allowed a team of seven Cure53 testers to audit the software system for a total of eighteen days. As a result of this wide-scoping and well-planned investigation, a good coverage of the Linkerd components was achieved. Cure53 is happy to report that no real vulnerabilities could be identified on the Linkerd scope. For the sake of completeness, the testing team should mention that there were several general security-relevant shortcomings. However, the development team was already well-aware of the issues raised by Cure53, i.e. as regards the necessary accessibility of the control plane from within all other containers linked to the data plane, which could, arguably, be abused by an untrusted application in the same space. Most importantly, work is already under way as far as tackling this is concerned. The Linkerd will handle the problem by introducing RBAC logic with the next major release. Another security-relevant aspect entails the component dependencies which have not been properly audited. Especially the currently used TLS library has not been examined at all and this should be rectified as a matter of urgency, especially since it is related to a particularly precarious area.

The general indicators of security found on the Linkerd project during this June 2019 assessment are all very good. Cure53 needs to mention the atypically excellent code readability, careful choice of implementation languages, as well as the clearly written and well-maintained documentation for all attributes. These aspects contribute to the body of evidence about the overall exceptional quality of the project in terms of security. The involved development team was highly engaged, starting from the preparation of this audit, to the deployment of the test cluster, and, finally, to quick reaction times as regards questions posed by the testers. The overall state of the Linkerd project - from a technical perspective and the in-house team's great awareness of security-relevant practices and aspect, solidly places the Linkerd complex on a very good level.

Cure53 would like to thank Oliver Gould and William Morgan from the Linkerd development 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 also needs to be extended to The Linux Foundation for sponsoring this project.