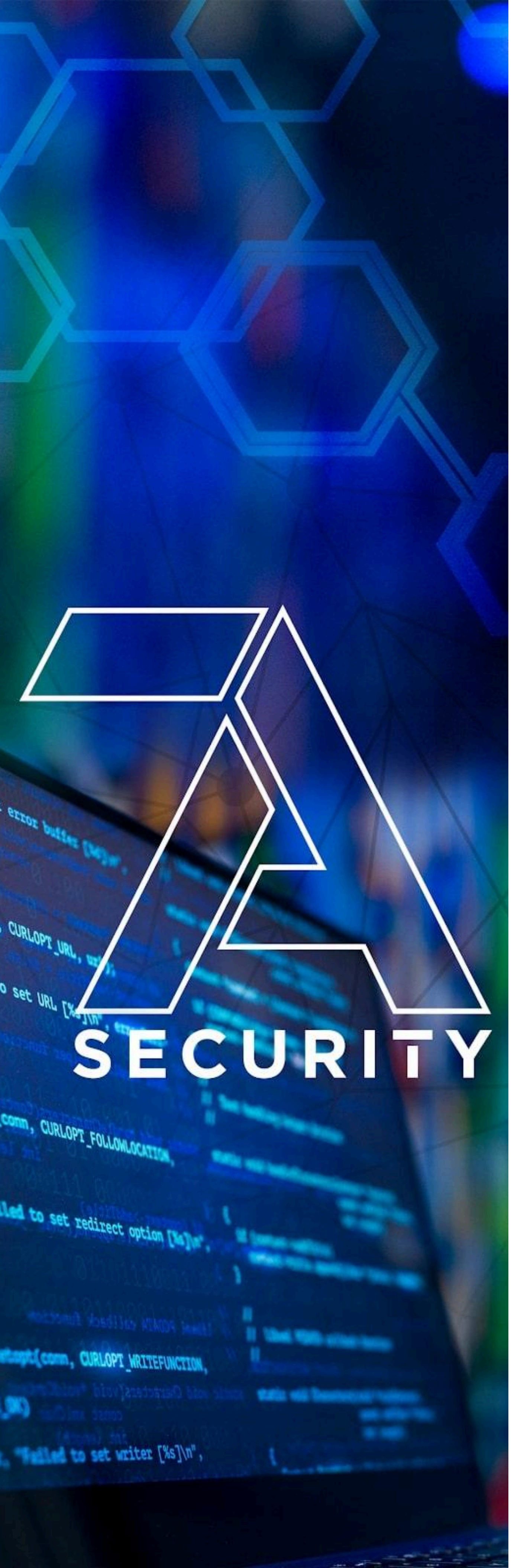

Test Targets:

Linkerd2 Main

Linkerd2 Proxy APIs

Pentest Report

Client:

Linkerd2 team

in collaboration with the

Open Source Technology Improvement Fund, Inc

7ASecurity Test Team:

- Abraham Aranguren, MSc.
- Daniel Ortiz, MSc.
- Miroslav Štampar, PhD.

7ASecurity

Protect Your Site & Apps
From Attackers

sales@7asecurity.com

7asecurity.com

INDEX

Introduction

> *"Linkerd is an ultralight, security-first service mesh for Kubernetes. Linkerd adds critical security, observability, and reliability features to your Kubernetes stack with no code change required."*
>
> From https://github.com/linkerd/linkerd2

This document outlines the results of a penetration test and *whitebox* security review conducted against the Linkerd2 platform. The project was solicited by Linkerd2, facilitated by the *Open Source Technology Improvement Fund, Inc (OSTIF)*, funded by the *Cloud Native Computing Foundation (CNCF)*, and executed by 7ASecurity from September until November 2024. The audit team dedicated 28 working days to complete this assignment. Please note that this is the third penetration test for this project. Consequently, the identification of security weaknesses was expected to be more difficult during this engagement, as more vulnerabilities are identified and resolved after each testing cycle.

During this iteration the goal was to review the solution as thoroughly as possible, to ensure Linkerd2 users can be provided with the best possible security. The methodology implemented was *whitebox*: 7ASecurity was provided with access to a staging environment, documentation, test users, and source code. A team of three senior auditors carried out all tasks required for this engagement, including preparation, delivery, documentation of findings and communication.

A number of necessary arrangements were in place by September 2024, to facilitate a straightforward commencement for 7ASecurity. In order to enable effective collaboration, information to coordinate the test was relayed through email, as well as a shared Slack channel. The Linkerd2 team was helpful and responsive throughout the audit, which ensured that 7ASecurity was provided with the necessary access and information at all times, thus avoiding unnecessary delays. 7ASecurity provided regular updates regarding the audit status and its interim findings during the engagement.

The findings of the security audit can be summarized as follows:

Identified Vulnerabilities	*Hardening Recommendations*	*Total Issues*
1	*6*	*7*

Moving forward, the scope section elaborates on the items under review, while the findings section documents the identified vulnerabilities followed by hardening recommendations with lower exploitation potential. Each finding includes a technical description, a proof-of-concept (PoC) and/or steps to reproduce if required, plus mitigation or fix advice for follow-up actions by the development team.

Finally, the report culminates with a conclusion providing detailed commentary, analysis, and guidance relating to the context, preparation, and general impressions gained throughout this test, as well as a summary of the perceived security posture of the Linkerd2 applications.

Scope

The following list outlines the items in scope for this project:

- **Linkerd2 Main repository:**
  - https://github.com/linkerd/linkerd2
- **Linkerd2 Proxy APIs:**
  - Linkerd2-proxy:
    - https://github.com/linkerd/linkerd2-proxy
  - Linkerd2-proxy-api
    - https://github.com/linkerd/linkerd2-proxy-api
  - Linkerd2-proxy-init
    - https://github.com/linkerd/linkerd2-proxy-init

Identified Vulnerabilities

This area of the report enumerates findings that were deemed to exhibit greater risk potential. Please note these are offered sequentially as they were uncovered, they are not sorted by significance or impact. Each finding has a unique ID (i.e. *LNK-01-001*) for ease of reference, and offers an estimated severity in brackets alongside the title.

LNK-01-003 Possible RCE via clear-text HTTP Instructions in Script *(High)*

Retest Notes: The Linkerd2 team resolved this issue[1] and 7ASecurity confirmed that the fix is valid.

The Linkerd2 codebase includes a *protoc* shell script that directs users to download the *unzip* utility from a clear-text HTTP URL. Furthermore, even if that URL was accessed over TLS, the top three links to download *unzip* on that page are clear-text FTP URLs. A malicious attacker able to modify clear-text HTTP communications (i.e. via Public Wi-Fi without guest isolation, DNS rebinding, ISP MitM, BGP Hijacking) could leverage this weakness to modify the legitimate *unzip* utility with an attacker-controlled binary, in situations where it is not already installed on the system.

Affected File:
https://github.com/linkerd/linkerd2/blob/[...]/bin/protoc

Affected Code:

```
require_from() {
    if ! command -v "$1" >/dev/null 2>/dev/null ; then
        echo "Please acquire $1 from $2" >&2
        return 1
    fi
}

if [ ! -f "$protocbin" ]; then
    require_from curl 'https://curl.se/download.html'
    require_from unzip 'http://infozip.sourceforge.net/UnZip.html#Downloads'
  [...]
fi
```

Please note that the top three links to download *unzip* from that sourceforge.net page are clear-text FTP links:

Affected URL:
https://infozip.sourceforge.net/UnZip.html#Downloads

[1] https://github.com/linkerd/linkerd2/commit/08a6dba655e99cd3a055711c16bd0334dd175a9f

Affected Content:
Ready-to-run binary versions of UnZip are available for numerous platforms and operating systems, but for most systems, only older binaries are available. The three primary CTAN sites (and their many mirrors) contain a snapshot of these binaries, current as of roughly 2004 (i.e., UnZip 5.51 and Zip 2.3 timeframe):

- *tug.ctan.org (US) [FROZEN]*
- *ftp.tex.ac.uk (UK) [FROZEN]*
- *ftp.dante.de (Germany) [FROZEN]*

As can be seen above, the proposed page to download *unzip* from starts with three clear-text FTP mirrors:
ftp://tug.ctan.org/tex-archive/tools/zip/info-zip/
ftp://ftp.tex.ac.uk/tex-archive/tools/zip/info-zip/
ftp://ftp.dante.de/tex-archive/tools/zip/info-zip/

This may be further validated with the clear-text telnet tool, connecting to port 21 (FTP) to any of the aforementioned servers:

Command:
```
telnet tug.ctan.org 21
```

Output:
```
Trying 155.101.98.136...
Connected to ftp.math.utah.edu.
Escape character is '^]'.
220 ftp.math.utah.edu FTP server ready.
```

The script should be updated to provide safer instructions for users to download the *unzip* utility. This should be an https:// URL, without any links to clear-text download resources, such as clear-text FTP mirrors. Incorporating secure download practices will enhance the security posture of Linkerd2, protecting users from potential MITM and integrity risks.

Hardening Recommendations

This area of the report provides insight into less significant weaknesses that might assist adversaries in certain situations. Issues listed in this section often require another vulnerability to be exploited, need an uncommon level of access, exhibit minor risk potential on their own, and/or fail to follow information security best practices. Nevertheless, it is recommended to resolve as many of these items as possible to improve the overall security posture and protect users in edge-case scenarios.

LNK-01-001 Enhanced Security Against MitM via TLS MinVersion *(Info)*

Retest Notes: The Linkerd2 team resolved this issue[2] and 7ASecurity confirmed that the fix is valid.

The Linkerd2 codebase currently supports TLS 1.2, but upgrading to TLS 1.3 is advised for enhanced security. Although TLS 1.2 is reliable and widely used, it is vulnerable to certain cryptographic weaknesses and attacks[3]. Starting in 2024, enforcing TLS 1.3[4] as the minimum version is recommended, due to greater security, widespread support, and six-year availability. Exceptions may be made for legacy clients needing older TLS versions. The issue originates from the following files:

Affected Files:
https://github.com/linkerd/linkerd2/blob/[...]/controller/webhook/server.go
https://github.com/linkerd/linkerd2/blob/[..]/viz/tap/api/server.go

Example Code:

```
func NewServer(
  [...]

   server := &http.Server{
      Addr:              addr,
      ReadHeaderTimeout: 15 * time.Second,
      TLSConfig: &tls.Config{
         MinVersion: tls.VersionTLS12,
      },
   }
```

While the likelihood of Man-In-The-Middle (MitM) attacks on Linkerd2 users is low, security can be further enhanced by configuring TLS instances to use *tls.VersionTLS13* as the minimum version.

[2] https://github.com/linkerd/linkerd2/commit/3847f9cf13c63f4829eebb7a81d461319ddc9261
[3] https://www.cloudflare.com/learning/ssl/why-use-tls-1.3/
[4] https://www.vertexcybersecurity.com.au/tls1-2-end-of-life/

LNK-01-002 Possible DYLIB Injection on MacOS Client *(Medium)*

The MacOS Linkerd2 CLI[5] is susceptible to DYLIB Injection attacks[6] due to a missing __*RESTRICT* segment and lack of a hardened runtime in the Mach-O file. A malicious attacker, able to set environment variables, might exploit this to inject dynamic libraries into a legitimate Linkerd2 process. These injected libraries could then execute arbitrary code within the process, potentially leading to unauthorized access, data theft, or system compromise.

To confirm this weakness it is necessary to compile a DYLIB library and use the *DYLD_INSERT_LIBRARIES* environment variable as shown in the following steps:

Step 1: Create a DYLIB Library to Inject

PoC Code:

```
#include <stdio.h>
#include <syslog.h>
__attribute__((constructor))

static void myconstructor(int argc, const char **argv)
{
printf("[+] dylib constructor called from %s\n", argv[0]);
syslog(LOG_ERR, "[+] dylib constructor called from %s\n", argv[0]);
}
```

Step 2: Compile the dynamic library

Command:

```
gcc -dynamiclib libtest.c -o libtest.dylib
```

Step 3: Inject the DYLIB library into the target application

Command:

```
DYLD_INSERT_LIBRARIES=libtest.dylib linkerd –help
```

Output:

```
[+] dylib constructor called from linkerd
linkerd manages the Linkerd service mesh.
Usage:
  linkerd [command]
Available Commands:
  authz         List authorizations for a resource
  check         Check the Linkerd installation for potential problems
```

[5] https://github.com/linkerd/linkerd2/[...]/edge-24.10.5/linkerd2-cli-edge-24.10.5-darwin-arm64
[6] https://attack.mitre.org/techniques/T1574/006/

```
[..]
```

This can also be confirmed by searching the desired string in the log stream:

Command:

```
log stream --style syslog --predicate 'eventMessage CONTAINS[c] "constructor"'
```

Output:

```
Filtering the log data using "composedMessage CONTAINS[c] "constructor""
Timestamp                       (process)[PID]
2024-11-04 22:28:03.493990-0300  localhost linkerd-edge-24.10.5[70969]: (libtest.dylib)
[+] dylib constructor called from linkerd
```

To mitigate DYLIB injection risks associated with the *DYLD_INSERT_LIBRARIES* environment variable on MacOS, a restricted segment should be enabled to prevent dynamic loading of *dylib* libraries for arbitrary code injection. It is recommended to use the following compiler options to enable the restricted segment feature:

Proposed fix 1 (compiler options on binaries that use DYLB linker):

```
-Wl,-sectcreate,__RESTRICT,__restrict,/dev/null
```

Alternatively, a hardened runtime entitlement[7] could be set on the Mach-O binary, please notice that this will require a paid subscription:

Proposed fix 2 (hardened runtime entitlement):

Command:

```
codesign -s cert --option=runtime linkerd
```

Command (check for hardened options):

```
codesign -dv linkerd
```

Output:

```
Executable=/Users/[...]/linkerd2
Identifier=main
Format=Mach-O thin (arm64)
CodeDirectory     v=20500     size=490437     flags=0x10000(runtime)     hashes=15321+2
location=embedded
Signature size=1644
Signed Time=4 Nov 2024 at 22:24:07
Info.plist=not bound
TeamIdentifier=not set
Runtime Version=11.0.0
Sealed Resources=none
Internal requirements count=1 size=80
```

[7] https://developer.apple.com/documentation/security/hardened_runtime

LNK-01-004 Possible WebSocket Hijacking via missing Validation *(Low)*

Retest Notes: The Linkerd2 team resolved this issue[8] and 7ASecurity confirmed that the fix is valid.

The Linkerd2 codebase is potentially vulnerable to *Cross-Site WebSocket Hijacking (CSWSH)*[9] due to missing validation of the Origin header during WebSocket handshakes. CSWSH, similar to Cross-Site Request Forgery (CSRF), exploits the WebSocket protocol to initiate connections from untrusted origins. Without proper Origin header validation, a malicious website could establish a WebSocket connection with the server, potentially enabling unauthorized actions on behalf of an authenticated user. This can be confirmed by observing the following code snippet:

Affected File:
https://github.com/linkerd/linkerd2/blob/[...]/web/srv/api_handlers.go

Affected Code:

```
func (h *handler) handleAPITap(w http.ResponseWriter, req *http.Request, p
httprouter.Params) {
    ws, err := websocketUpgrader.Upgrade(w, req, nil)
    if err != nil {
        renderJSONError(w, err, http.StatusInternalServerError)
        return
    }
    defer ws.Close()
```

According to the Gorilla WebSocket documentation[10], if the *CheckOrigin* function is *nil*, requests are denied when the *Origin* header is present but does not match the *Host* header, providing basic validation. This behavior was confirmed at runtime by 7ASecurity, where exploitation attempts failed under these conditions. However, relying on the default behavior of a third-party library introduces unnecessary risks.

To mitigate these risks, it is recommended to explicitly implement a *CheckOrigin* function in *websocketUpgrader* to validate that the *Origin* header matches trusted domains. This ensures only trusted origins can establish WebSocket connections[11], providing an additional layer of security.

[8] https://github.com/linkerd/linkerd2/commit/28c1b7807744caaf3a36950a3d004e76b0e38ab1
[9] https://portswigger.net/web-security/websockets/cross-site-websocket-hijacking
[10] https://pkg.go.dev/github.com/gorilla/websocket#Upgrader.CheckOrigin
[11] https://stackoverflow.com/a/65039729

LNK-01-005 Possible MitM via Insecure gRPC without TLS *(Info)*

Note: It was later found that while *insecure.NewCredentials()* allows the bypass of TLS, for local development or non-production systems, Linkerd2 will still employ a layer of encryption at a higher level in all relevant scenarios, such as CLI to Control Plane and Proxy to Control Plane. Nevertheless, appropriate security warnings ought to be in place to alert customers when they disable encryption layers in Linkerd2 applications.

While Linkerd2 automatically enforces mutually authenticated *Transport Layer Security (mTLS)* for all TCP traffic among interconnected pods[12], Linkerd does not require mTLS unless authorization policies are configured[13]. Therefore, given certain gRPC clients and servers use *insecure.NewCredentials()*[14], which disables transport security, malicious attackers may be able to intercept, eavesdrop, and manipulate transmitted data in at least some configurations. This can be confirmed by observing the following code snippets:

Affected Files:
https://github.com/linkerd/linkerd2/blob[..]/cli/cmd/policy.go#L92
https://github.com/linkerd/linkerd2/blob/[...]/controller/api/destination/client.go#L21
https://github.com/linkerd/linkerd2/blob[..]/viz/metrics-api/client/client.go#L51
https://github.com/linkerd/linkerd2/blob/[..]/viz/tap/api/client.go#L11
https://github.com/linkerd/linkerd2/blob/[..]/viz/tap/api/grpc_server.go#L311

Affected Code:

```
func NewClient(addr string) (pb.DestinationClient, *grpc.ClientConn, error) {
   conn, err := grpc.NewClient(addr,
grpc.WithTransportCredentials(insecure.NewCredentials()),
      grpc.WithStatsHandler(&ocgrpc.ClientHandler{}))
   if err != nil {
      return nil, nil, err
   }

   return pb.NewDestinationClient(conn), conn, nil
}
```

To ensure secure and encrypted gRPC transport, avoid *insecure.NewCredentials()*, as it lacks authentication and encryption. Use *credentials.NewClientTLSFromFile()* to enable TLS, ensuring all client-server data is encrypted and authenticated, thus reducing the risk of unauthorized access and eavesdropping.

[12] https://linkerd.io/2.16/features/automatic-mtls/
[13] https://linkerd.io/2.16/features/automatic-mtls/#caveats-and-future-work
[14] https://pkg.go.dev/google.golang.org/grpc/credentials/insecure#NewCredentials

LNK-01-006 Multiple Vulnerabilities Found in Docker Images *(Low)*

Retest Notes: The Linkerd2 team resolved this issue[15] and 7ASecurity confirmed that the fix is valid.

Several Docker images contain components with publicly known vulnerabilities, introducing security risks. As these vulnerabilities are contained within the Docker containers and do not directly impact the host system, the severity is reduced. The tables below summarize significant vulnerabilities in the outdated images.

Issue 1: Docker image rust:1.76.0[16]

Affected Files:
https://github.com/linkerd/linkerd2/blob/[...]/.github/workflows/codecov.yml
https://github.com/linkerd/linkerd2-proxy/blob/[...]/.github/workflows/fuzzers.yml

Affected Code:

```
rust:
  name: Rust
  runs-on: ubuntu-22.04
  timeout-minutes: 15
  container:
    image: docker://rust:1.76.0
```

Vulnerable Components:

Component	Issue	Severity
libexpat	CVE-2024-45492[17] - *nextScaffoldPart* in *xmlparse.c* can have an integer overflow	Critical
	CVE-2024-45491[18] - *dtdCopy* in *xmlparse.c* can have an integer overflow for nDefaultAtts	
	CVE-2024-45490[19] - *xmlparse.c* does not reject a negative length for *XML_ParseBuffer*	High
git	CVE-2024-32002[20] - Repositories with submodules	Critical

[15] https://github.com/linkerd/linkerd2-proxy-init/pull/450
[16] https://hub.docker.com/layers/library/rust/1.76.0/images/sha256-c3a[...]ccf
[17] https://scout.docker.com/vulnerabilities/id/CVE-2024-45492
[18] https://scout.docker.com/vulnerabilities/id/CVE-2024-45491
[19] https://scout.docker.com/vulnerabilities/id/CVE-2024-45490
[20] https://scout.docker.com/vulnerabilities/id/CVE-2024-32002

	can be crafted in a way that exploits a bug in Git whereby it can be fooled into writing files not into the submodule worktree but into a *.git/* directory	
	CVE-2024-32004[21], CVE-2024-32465[22] - Specially crafted local repository, when cloned, will execute arbitrary code during the operation	High
	CVE-2023-29007[23] - Specially crafted *.gitmodules* file with submodule URLs that are longer than 1024 characters can used to exploit a bug in *config.c::git_config_copy_or_rename_section_in_file()*	
	CVE-2023-25652[24] - Specially crafted input to *git apply --reject*, a path outside the working tree can be overwritten with partially controlled contents	
krb5	CVE-2024-37371[25] - Invalid memory read in GSS token handling	Critical
	CVE-2024-37370[26] - Potential plaintext modification in confidential GSS krb5 wrap token	High
libaom	CVE-2024-5171[27] - Integer overflow in *libaom* internal function *img_alloc_helper* can lead to heap buffer overflow	Critical
glibc	CVE-2024-33602[28] - *netgroup* cache can corrupt memory	High
	CVE-2024-33601[29] - *netgroup* cache may terminate daemon on memory allocation failure	
	CVE-2024-2961[30] - *iconv()* function may overflow	

[21] https://scout.docker.com/vulnerabilities/id/CVE-2024-32004
[22] https://scout.docker.com/vulnerabilities/id/CVE-2024-32465
[23] https://scout.docker.com/vulnerabilities/id/CVE-2023-29007
[24] https://scout.docker.com/vulnerabilities/id/CVE-2023-25652
[25] https://scout.docker.com/vulnerabilities/id/CVE-2024-37371
[26] https://scout.docker.com/vulnerabilities/id/CVE-2024-37370
[27] https://scout.docker.com/vulnerabilities/id/CVE-2024-5171
[28] https://scout.docker.com/vulnerabilities/id/CVE-2024-33602
[29] https://scout.docker.com/vulnerabilities/id/CVE-2024-33601
[30] https://scout.docker.com/vulnerabilities/id/CVE-2024-2961

	the output buffer	

Issue 2: Docker image alpine:3.19.0[31]

Affected File:
https://github.com/linkerd/linkerd2-proxy-init/…/integration/linkerd-cni-config.yml

Affected Code:

```
extraInitContainers:
- name: sleep
  image: alpine:3.19.0
  command: ["/bin/sh", "-c", "sleep 15"]
```

Vulnerable Components:

Component	Issue	Severity
openssl	CVE-2024-5535[32] - Calling the OpenSSL API function *SSL_select_next_proto()* with an empty supported client protocols buffer may cause a crash or memory contents to be sent to the peer	Critical
	CVE-2024-6119[33] - Applications performing certificate name checks (e.g., TLS clients checking server certificates) may attempt to read an invalid memory address resulting in abnormal termination of the application process	High

It is recommended to update the affected Docker images to the latest stable versions to mitigate these vulnerabilities. Avoid sticking to specific minor revisions (e.g., *alpine:3.19.0*) and ensure that the images are regularly updated to incorporate security patches. Additionally, consider using automated security scanning tools such as *Trivy*[34] or *Snyk*[35] to identify and address vulnerabilities in Docker images on an ongoing basis.

[31] https://hub.docker.com/layers/library/alpine/3.19.0/images/sha256-13b[...]bcd
[32] https://scout.docker.com/vulnerabilities/id/CVE-2024-5535
[33] https://scout.docker.com/vulnerabilities/id/CVE-2024-6119
[34] https://github.com/aquasecurity/trivy
[35] https://docs.snyk.io/scan-with-snyk/snyk-container/scan-container-images

LNK-01-007 Multiple Vulnerable Dependencies *(Low)*

It was established that the Linkerd2 codebase makes use of components with publicly known vulnerabilities. While most of these weaknesses are likely not exploitable under the current implementation, this is still a bad practice that could result in unwanted security vulnerabilities. The following table summarizes the publicly known vulnerabilities affecting packages used either directly or as an underlying dependency.

Component	Issues	Severity
micromatch <4.0.8	Regular Expression Denial of Service[36]	Medium
send <0.19.9	Template Injection[37]	Medium
http-proxy-middleware <2.0.7	Denial of Service[38]	High
cookie <0.7.0	Cookies with Out-of-Bound Characters[39]	High

This issue was confirmed by reviewing the following file:

Affected File:
https://github.com/linkerd/linkerd2/blob/[...]/web/app/yarn.lock

Affected Code:
```
micromatch@4.0.2:
  version "4.0.2"
  resolved
"https://registry.yarnpkg.com/micromatch/-/micromatch-4.0.2.tgz#4fcb0999bf9fbc2fcbdd212
f6d629b9a56c39259"
[...]
send@0.18.0:
  version "0.18.0"
  resolved
"https://registry.yarnpkg.com/send/-/send-0.18.0.tgz#670167cc654b05f5aa4a767f9113bb371b
c706be"
[...]
http-proxy-middleware@^2.0.3:
  version "2.0.4"
  resolved
"https://registry.yarnpkg.com/http-proxy-middleware/-/http-proxy-middleware-2.0.4.tgz#0
3af0f4676d172ae775cb5c33f592f40e1a4e07a"
```

[36] https://www.npmjs.com/advisories/1098681
[37] https://www.npmjs.com/advisories/1099525
[38] https://www.npmjs.com/advisories/1100223
[39] https://www.npmjs.com/advisories/1099846

```
[...]
cookie@0.6.0:
  version "0.6.0"
  resolved
"https://registry.yarnpkg.com/cookie/-/cookie-0.6.0.tgz#2798b04b071b0ecbff0dbb62a505a8e
fa4e19051"
```

It is recommended to upgrade all underlying dependencies to their current versions to resolve the above issues. To prevent similar issues in the future, an automated task or commit hook should be implemented to regularly check for vulnerabilities in dependencies. Suggested solutions include *yarn audit*[40], the *Snyk* tool[41] and the *OWASP Dependency Check* project[42]. Ideally, such tools should be run regularly by an automated job (e.g. in CI/CD pipeline) that alerts a lead developer or administrator about known vulnerabilities in dependencies so that the patching process can start in a timely manner.

[40] https://classic.yarnpkg.com/lang/en/docs/cli/audit/
[41] https://snyk.io/
[42] https://owasp.org/www-project-dependency-check/

Conclusion

The Linkerd2 solution defended itself well against a broad range of attack vectors. The platform will become increasingly difficult to attack as additional cycles of security testing and subsequent hardening continue.

The Linkerd2 platform provided a number of positive impressions during this assignment that must be mentioned here:

- Proactive security practices are in place, such as utilizing SAST tools like Gosec and fuzzing tests, as well as using linters, and automated dependency updates in GitHub workflows.
- Other proactive security measures have been clearly implemented, such as active evaluation of SAST tool flags, well-reasoned configuration justifications, and an actively maintained and well-audited codebase with a focus on addressing security risks.
- The Linkerd2 codebase demonstrates its source code is robust and has been thoroughly audited and enhanced multiple times, which makes uncovering new security weaknesses particularly challenging, and highlights the value of regular penetration testing and hardening iterations.
- No major vulnerabilities could be found during the engagement, with only a minor exception noted in an outdated script that was later removed during the test (LNK-01-003).

The Linkerd2 solution would benefit from addressing the following security issues:

- **Enhancing Security with TLS 1.3:** Enforce TLS 1.3 as the minimum version, to mitigate cryptographic weaknesses in TLS 1.2, and align with modern standards (LNK-01-001).
- **Mitigation of DYLIB Injection:** Implementing a hardened runtime, including the *__RESTRICT* segment in the *Mach-O* file for the *macOS* Linkerd2 CLI, will prevent malicious library injections, reducing the odds of unauthorized code execution and system compromise in edge-case scenarios (LNK-01-002).
- **Default mTLS Enforcement:** Enforcing mTLS by default, regardless of configured authorization policies, will ensure secure communication and further mitigate Man-In-The-Middle (MitM) attacks against gRPC clients and servers (LNK-01-005).
- **Software Patching:** The Linkerd2 solution should implement appropriate software patching procedures which regularly apply security patches in a timely manner, this applies to Docker images (LNK-01-006), as well as the Linkerd2 project itself (LNK-01-007). In a day and age when most lines of code come from underlying software dependencies, regularly patching these becomes increasingly important to avoid unwanted security vulnerabilities. Possible

automation for this could include tools like *Snyk.io*[43] or *Renovate Bot*[44].

It is advised to address all issues identified in this report, including informational and low severity tickets where possible. This will not just strengthen the security posture of the application significantly, but also reduce the number of tickets in future audits.

Once all issues in this report are addressed and verified, a more thorough review, ideally including another source code audit, is highly recommended to ensure adequate security coverage of the platform. This provides auditors with an edge over possible malicious adversaries that do not have significant time or budget constraints.

Please note that future audits should ideally allow for a greater budget so that test teams are able to deep dive into more complex attack scenarios. Some examples of this could be third party integrations, complex features that require to exercise all the application logic for full visibility, authentication flows, challenge-response mechanisms implemented, subtle vulnerabilities, logic bugs and complex vulnerabilities derived from the inner workings of dependencies in the context of the application. Additionally, the scope could perhaps be extended to include other internet-facing Linkerd2 resources.

It is suggested to test the application regularly, at least once a year or when substantial changes are going to be deployed, to make sure new features do not introduce undesired security vulnerabilities. This proven strategy will reduce the number of security issues consistently and make the application highly resilient against online attacks over time.

7ASecurity would like to take this opportunity to sincerely thank David McLaughlin, William Morgan and the rest of the Linkerd2 team, for their exemplary assistance and support throughout this audit. Last but not least, appreciation must be extended to the Open Source Technology Improvement Fund (OSTIF) for facilitating and managing this project, and thank you to CNCF for funding the effort.

[43] https://snyk.io/
[44] https://github.com/renovatebot/renovate