PRESENTS

Fuzzing integration of Linkerd2-proxy and dependencies

in collaboration with

Authors

David Korczynski <david@adalogics.com>
Adam Korczynski <adam@adalogics.com>
Date: 22nd April, 2021

Executive summary

The overall goal of the engagement described in this report was to integrate security and reliability analysis by way of fuzzing into the Linkerd2-proxy project as well as its dependencies. This was done in a manner such that vulnerability analysis will happen continuously (even after the engagement). The source code of Linkerd2-proxy is written in Rust and this makes it particularly vulnerable to unknown panics.

Scope of engagement

The project ran from 8th of March to and including 16th of April and the entire scope of the Linkerd2-proxy project was considered.

Methodology

Ada Logic's researchers performed an initial analysis of Linkerd2-proxy to understand the potential of integrating fuzzing into the project as well as identify the external dependencies that are suitable for fuzzing. Following this, Ada Logics entered a phase of developing fuzzers for Linkerd2-proxy as well as dependencies, and also integrate the various projects into Google's OSS-Fuzz service. The effect of this is that the fuzzers run continuously both during and after this engagement. The results of the engagement can be summarised as follows:

Results summarised
8 projects integrated into OSS-Fuzz (5 accepted, 2 pending, 1 rejected)
14 fuzzers written in total. 7 fuzzers for Linkerd2-proxy, 7 for external dependencies.
2 bugs found by linkerd2-proxy fuzzers, and several bugs in external dependencies.
Code and fixes committed upstream

Engagement process and methodology

In this section we discuss the overall process during the engagement.

Integrate fuzzing into Linkerd2-proxy and relevant dependencies

The first step was to identify projects relevant for fuzzing in the linkerd2-proxy ecosystem. Given the time constraints of the project, the way we approached this was to identify the external dependencies of linkerd2-proxy and select projects that were most welcoming to fuzzing as well as of high usage in the Rust community in general. The result of our efforts was to integrate fuzzing into the following projects:

- prost: protobuf library for Rust
- Rustls: modern implementation of TLS in Rust
- hyper: A selection of libraries that use http implementations
    - h2
    - http
- html-escape
- unicode-* libraries
    - Unicode segmentation
    - Unicode normalisation
- httparse
- Linkerd2-proxy itself
- base64

In addition to this, some of the fuzzers that we wrote also target code in other projects, such as trust-dns, as we will see in the results section.

Several tasks were carried out on these projects depending on the state of the project. For example, some projects already had fuzzers integrated but were not integrated into OSS-Fuzz, whereas some projects had no no fuzzers in their project. For each project, we, therefore, performed one or both of the following tasks:

1. Write fuzzers for the project
2. Integrate the project into OSS-Fuzz

Work performed on the projects

The following summarises the tasks for each project and also provide links to various commits with the work performed:

Prost

- **Fuzzer written:** No
- **OSS-Fuzz integration:** Yes

- **Description of work:** The prost repository is a protobuf implementation in Rust. The project already had fuzzers integrated, so we integrated these fuzzers to be run continuously.
- **Relevant PRs**
    - https://github.com/google/oss-fuzz/pull/5404

Rustls

- **Fuzzer written:** Yes
- **OSS-Fuzz integration:** Yes
- **Description of work:** rustls already had fuzzers written. However, one of these were no longer runnable so we fixed this up and then integrated the project to be run continuously with OSS-Fuzz.
- **Relevant PRs:**
    - https://github.com/google/oss-fuzz/pull/5332

Hyperium

- **Fuzzer written:** Yes
- **OSS-Fuzz integration:** Yes
- **Description of work:** Hyperium is a collection of libraries used for creating network-based applications. They already had some fuzzers implemented, although these were out-dated. For this project we wrote 3 new fuzzers as well as fixed an existing fuzzer that were no longer able to run. We integrated two of the projects related to the hyperium framework into OSS-Fuzz.
- **Relevant PRs:**
    - https://github.com/google/oss-fuzz/pull/5330
    - https://github.com/hyperium/http/pull/478
    - https://github.com/hyperium/h2/pull/529

Unicode-rs

- **Fuzzer written:** Yes
- **OSS-Fuzz integration:** Yes
- **Description of work:** We integrated two of the projects related to the Unicode-rs framework into OSS-Fuzz. For one of these projects we wrote fuzzers whereas the other project already had fuzzers integrated.
- **Relevant PRs:**
    - https://github.com/google/oss-fuzz/pull/5413
    - https://github.com/unicode-rs/unicode-segmentation/pull/94

Httparse

- **Fuzzer written:** No
- **OSS-Fuzz integration:** Yes
- **Description of work:** httparse already had three fuzzers written. We integrated the project into OSS-Fuzz so these fuzzers are written continuously.
- **Relevant PRs:**
    - https://github.com/google/oss-fuzz/pull/5331
    - https://github.com/seanmonstar/httparse/pull/93

Linkerd2-proxy

- **Fuzzer written:** Yes
- **OSS-Fuzz integration:** Yes
- **Description of work:** Created 7 fuzzers for the Linkerd2-proxy that target various parts of the proxy. The fuzzers each follow a similar style in terms of set up, which makes it straightforward to create additional fuzzers by the Linkerd2-proxy team. Linkerd2-proxy was also integrated into OSS-Fuzz now.
- **Relevant PRs:**
  - **https://github.com/linkerd/linkerd2-proxy/pull/977**
  - **https://github.com/linkerd/linkerd2-proxy/pull/961**
  - **https://github.com/linkerd/linkerd2-proxy/pull/978**
  - **https://github.com/google/oss-fuzz/pull/5547**
  - **https://github.com/google/oss-fuzz/pull/5625**

Base64

- **Fuzzer written:** Yes
- **OSS-Fuzz integration:** Yes, but rejected.
- **Description of work:** Integrated a fuzzer for the Rust base64 implementation. However, the maintainers of OSS-Fuzz deemed this project too simple for continuous analysis so it was rejected by the OSS-Fuzz team for integration.
- **Relevant PRs:**
  - https://github.com/google/oss-fuzz/pull/5412

Html-escape

- **Fuzzer written:** Yes
- **OSS-Fuzz integration:** Yes
- **Description of work:** Fuzzers were written for the encoding and decoding functions in the html-escape project. However, maintainers of the project are still waiting to reply.
- **Relevant PRs:**
  - https://github.com/google/oss-fuzz/pull/5411

Linkerd2-proxy fuzzers

In this section we will go over the fuzzer set up that we created for the Linkerd2-proxy and some of the external dependencies. We will leave out details of the fuzzers written for external dependencies in this report. However, the links provided in the above section directs to the code of the fuzzers if further information is desired. The focus of this section is also to encapsulate how the Linkerd2-proxy team can progress forward. The code for each of the fuzzers has already been reviewed thoroughly by the Linkerd2-proxy maintainers so we will not go into detail with this here.
One important limitation to highlight in writing fuzzers for the Linkerd2-proxy project is that several crates cannot be compiled with nightly rustc. This is needed in order to compile the fuzzers. The most important crate is **linkerd/app/integration.** During the engagement we were unable to compile this and the details have been discussed with the Linkerd2-proxy

team. Essentially, rustc completely exhausts the memory (the compilation process is killed at 70GB of memory used).

The following fuzzers have been written for Linkerd2-proxy

Target crate	Fuzzer name	Fuzzer link
transport-header	fuzz_target_raw	https://github.com/linkerd/linkerd2-proxy/blob/b4017463a04b30aa6a761f4a9ff1ebc0fded8a42/linkerd/transport-header/fuzz/fuzz_targets/fuzz_target_raw.rs
transport-header	fuzz_target_structured	https://github.com/linkerd/linkerd2-proxy/blob/b4017463a04b30aa6a761f4a9ff1ebc0fded8a42/linkerd/transport-header/fuzz/fuzz_targets/fuzz_target_structured.rs
addr	fuzz_addr	https://github.com/linkerd/linkerd2-proxy/blob/b4017463a04b30aa6a761f4a9ff1ebc0fded8a42/linkerd/addr/fuzz/fuzz_targets/fuzz_target_1.rs
dns	fuzz_dns	https://github.com/linkerd/linkerd2-proxy/blob/e0c2abaef9d3743fd4e38e2be8fc8f1ce8ab9083/linkerd/dns/fuzz/fuzz_targets/fuzz_target_1.rs
proxy/http	fuzz_http	https://github.com/linkerd/linkerd2-proxy/blob/b4017463a04b30aa6a761f4a9ff1ebc0fded8a42/linkerd/proxy/http/fuzz/fuzz_targets/fuzz_target_1.rs
tls	fuzz_tls	https://github.com/linkerd/linkerd2-proxy/blob/b4017463a04b30aa6a761f4a9ff1ebc0fded8a42/linkerd/tls/fuzz/fuzz_targets/fuzz_target_1.rs
app/inbound	fuzz_inbound	https://github.com/linkerd/linkerd2-proxy/blob/3a3f25fa2ebcb4473eb6f21790e7c0e4f648cf3e/linkerd/app/inbound/fuzz/fuzz_targets/fuzz_target_1.rs

Results

In this section we will go over the results of the fuzzers so far.

Bugs found

The fuzzers have so far have found two bugs.

The first bug was due to a panic when domain search causes names to exceed 255 bytes. Dpcumentation on the issue is found here https://github.com/bluejekyll/trust-dns/issues/1447 and a fixed for the issue was created here https://github.com/bluejekyll/trust-dns/pull/1448

The second bug was a panic thrown in the Addr crate. The issue is discussed in detail here https://github.com/linkerd/linkerd2-proxy/pull/976 with a fix included.

Several bugs have been reported in the Hyperium projects, however, we will refrain from disclosing these bugs as the maintainers of Hyperium are not involved in this project. The bugs have been communicated with the Linkerd2-proxy maintainers who have also received access to the project on oss-fuzz.

No bugs were found in httparse and rustls.

Coverage of Linkerd2-proxy

The following table shows the coverage after running each fuzzer for 20 minutes:

PATH	LINE COVERAGE	FUNCTION COVERAGE	REGION COVERAGE
duplex/	0.00% (0/160)	0.00% (0/19)	0.00% (0/124)
errno/	0.00% (0/23)	0.00% (0/12)	0.00% (0/404)
error/	0.00% (0/3)	0.00% (0/1)	0.00% (0/1)
identity/	0.00% (0/270)	0.00% (0/66)	0.00% (0/139)
retry/	0.00% (0/3)	0.00% (0/1)	0.00% (0/1)
tracing/	0.00% (0/285)	0.00% (0/42)	0.00% (0/134)
trace-context/	7.75% (20/258)	12.50% (6/48)	4.74% (10/211)
opencensus/	8.33% (9/108)	5.26% (1/19)	1.43% (1/70)
service-profiles/	15.38% (86/559)	21.24% (24/113)	8.48% (33/389)
exp-backoff/	20.00% (13/65)	8.33% (1/12)	19.44% (7/36)
tls/	25.21% (120/476)	15.45% (17/110)	29.17% (105/360)
app/	27.42% (967/3527)	12.89% (132/1024)	12.48% (289/2316)
proxy/	29.79% (847/2843)	28.64% (167/583)	22.10% (372/1683)
io/	30.17% (54/179)	29.17% (14/48)	28.00% (21/75)
http-classify/	33.33% (5/15)	60.00% (3/5)	66.67% (4/6)
metrics/	33.56% (100/298)	29.17% (21/72)	20.33% (37/182)
drain/	34.69% (17/49)	26.67% (4/15)	16.67% (4/24)
timeout/	41.94% (52/124)	25.00% (6/24)	20.93% (18/86)
http-metrics/	43.14% (236/547)	41.67% (35/84)	31.36% (90/287)
dns/	43.23% (67/155)	40.43% (19/47)	34.11% (44/129)
reconnect/	53.25% (41/77)	100.00% (7/7)	50.00% (23/46)
stack/	59.13% (311/526)	54.40% (68/125)	36.46% (101/277)
addr/	60.57% (106/175)	48.15% (26/54)	63.20% (79/125)
http-box/	71.43% (85/119)	75.76% (25/33)	76.74% (33/43)
cache/	74.39% (61/82)	100.00% (12/12)	58.14% (25/43)
conditional/	75.00% (3/4)	75.00% (3/4)	75.00% (6/8)
error-metrics/	78.08% (57/73)	85.71% (12/14)	65.38% (17/26)
transport-header/	80.34% (143/178)	48.78% (20/41)	63.91% (85/133)
concurrency-limit/	91.49% (43/47)	87.50% (7/8)	75.76% (25/33)
error-respond/	100.00% (27/27)	100.00% (8/8)	100.00% (12/12)
TOTALS	30.83% (3470/11255)	24.07% (638/2651)	19.47% (1441/7403)

Finally, the coverage of external dependencies by way of the Linkerd2-proxy fuzzers are as follows (projects not shown have 0% coverage):

tonic-0.4.2/	1.03% (20/1940)	1.19% (5/419)	0.68% (6/879)
tokio-util-0.6.6/	1.16% (17/1465)	2.13% (4/188)	1.28% (8/623)
unicode-bidi-0.3.5/	2.43% (18/740)	5.88% (4/68)	3.46% (15/434)
log-0.4.14/	2.86% (12/420)	1.63% (2/123)	1.04% (2/193)
libc-0.2.93/	5.26% (4/76)	4.76% (1/21)	4.00% (1/25)
lock_api-0.4.3/	7.53% (84/1116)	9.84% (18/183)	5.28% (18/341)
webpki-0.21.4/	8.56% (105/1226)	6.85% (10/146)	7.34% (62/845)
rand_core-0.6.2/	9.29% (29/312)	9.43% (5/53)	11.45% (15/131)
rand-0.8.3/	9.83% (126/1282)	10.24% (21/205)	7.65% (44/575)
slab-0.4.3/	10.75% (10/93)	13.33% (2/15)	4.08% (2/49)
tracing-core-0.1.17/	15.19% (166/1093)	13.77% (38/276)	13.18% (73/554)
smallvec-1.6.1/	16.56% (125/755)	22.41% (26/116)	14.25% (51/358)
arbitrary-1.0.0/	17.11% (110/643)	12.82% (20/156)	14.32% (56/391)
futures-task-0.3.14/	17.12% (50/292)	10.98% (9/82)	10.98% (9/82)
hashbrown-0.9.1/	17.64% (438/2483)	15.25% (63/413)	17.96% (187/1041)
unicode-normalization-0.1.17/	18.49% (366/1979)	36.67% (33/90)	9.16% (209/2281)
mio-0.7.11/	20.68% (416/2012)	18.30% (82/448)	14.80% (155/1047)
once_cell-1.7.2/	20.94% (102/487)	16.48% (15/91)	21.83% (55/252)
getrandom-0.2.2/	21.46% (53/247)	21.95% (9/41)	14.71% (25/170)
parking_lot-0.11.1/	23.67% (335/1415)	21.23% (31/146)	19.41% (138/711)
prost-0.7.0/	24.88% (300/1206)	21.33% (32/150)	20.51% (112/546)
tracing-0.1.25/	27.65% (128/463)	24.05% (19/79)	29.64% (75/253)
resolv-conf-0.7.0/	27.83% (113/406)	25.37% (17/67)	18.23% (74/406)
trust-dns-proto-0.20.2/	27.88% (1734/6219)	26.18% (289/1104)	22.63% (877/3876)
tower-0.4.6/	30.37% (352/1159)	26.24% (53/202)	25.32% (137/541)
hyper-0.14.7/	31.64% (2351/7430)	30.48% (313/1027)	26.45% (1216/4598)
http-0.2.4/	36.48% (1463/4010)	31.47% (231/734)	42.24% (1616/3826)
bytes-1.0.1/	38.70% (736/1902)	27.25% (100/367)	33.24% (243/731)
indexmap-1.6.2/	39.29% (132/336)	38.24% (26/68)	37.70% (46/122)
futures-util-0.3.14/	40.63% (629/1548)	32.96% (88/267)	34.74% (238/685)
httpdate-1.0.0/	41.18% (91/221)	6.90% (2/29)	22.96% (31/135)
futures-channel-0.3.14/	44.19% (430/973)	32.21% (48/149)	38.51% (191/496)
tokio-1.5.0/	45.05% (6196/13753)	37.82% (866/2290)	39.01% (2420/6203)
ppv-lite86-0.2.10/	45.29% (173/382)	37.65% (32/85)	38.02% (46/121)
signal-hook-registry-1.3.0/	48.35% (161/333)	37.21% (16/43)	32.54% (41/126)
untrusted-0.7.1/	49.60% (62/125)	39.53% (17/43)	45.12% (37/82)
http-body-0.4.1/	50.00% (35/70)	60.00% (9/15)	61.11% (11/18)
pin-project-1.0.7/	50.00% (5/10)	50.00% (1/2)	50.00% (1/2)
rand_chacha-0.3.0/	50.36% (138/274)	34.88% (15/43)	42.86% (33/77)
want-0.3.0/	54.78% (86/157)	40.00% (10/25)	62.86% (44/70)
trust-dns-resolver-0.20.2/	57.73% (1285/2226)	48.83% (146/299)	49.64% (476/959)
num_cpus-1.13.0/	63.04% (116/184)	80.00% (20/25)	62.14% (87/140)
linked-hash-map-0.5.4/	63.84% (113/177)	76.92% (20/26)	67.57% (50/74)
itoa-0.4.7/	66.67% (50/75)	83.33% (5/6)	72.41% (21/29)
lru-cache-0.1.2/	67.57% (25/37)	60.00% (6/10)	61.54% (8/13)
try-lock-0.2.3/	71.05% (27/38)	75.00% (6/8)	61.54% (8/13)
httparse-1.4.0/	73.64% (461/626)	60.81% (45/74)	63.89% (253/396)
libfuzzer-sys-0.4.0/	76.32% (29/38)	80.00% (4/5)	73.33% (11/15)
parking_lot_core-0.8.3/	77.17% (703/911)	69.14% (56/81)	70.05% (269/384)
idna-0.2.3/	80.89% (525/649)	63.46% (33/52)	78.69% (325/413)
pin-project-lite-0.2.6/	80.95% (51/63)	83.33% (5/6)	72.22% (13/18)
tower-layer-0.3.1/	85.19% (23/27)	81.82% (9/11)	81.82% (9/11)
futures-core-0.3.14/	87.04% (94/108)	70.00% (7/10)	74.29% (26/35)
lazy_static-1.4.0/	88.89% (32/36)	83.33% (5/6)	81.82% (9/11)
tinyvec-1.2.0/	89.60% (293/327)	97.50% (39/40)	76.19% (96/126)
hostname-0.3.1/	94.59% (35/37)	80.00% (4/5)	80.00% (8/10)
dyn-clone-1.0.4/	100.00% (14/14)	100.00% (2/2)	100.00% (6/6)
fnv-1.0.7/	100.00% (14/14)	100.00% (3/3)	100.00% (8/8)
tinyvec_macros-0.1.0/	100.00% (5/5)	100.00% (1/1)	100.00% (5/5)
tower-service-0.3.1/	100.00% (9/9)	100.00% (3/3)	100.00% (3/3)
TOTALS	14.58% (21805/149600)	13.65% (3001/21991)	12.94% (10311/79681)

Advice following engagement

This work performs a significant first step in integrating fuzzers into the ecosystem of Linkerd2-proxy. However, linkerd2-proxy is a complex system itself and has a large set of dependencies. In this section we will outline the advice we have on how the linkerd2-proxy developers can continue to progress with fuzzer integrations.

Short-term advice

1. Ensure that integration tests can be compiled with cargo fuzz. Then more end-to-end fuzzers (similar to fuzz_inbound) should be created based on the integration tests.
2. Linkerd2-proxy is part of a larger infrastructure involving Linkerd2. Linkerd2 should also be covered in fuzzers. We draw a reference here to the Envoy-proxy which has a total of 51 fuzzers integrated as a means of comparison.

Long-term advice

1. Our long-term advice it to continuously iterate the process
    a. Assess which parts of the code is missing fuzzing coverage by looking at code coverage output.
    b. Create fuzzers that have potential to reach this code.
    c. Let the new fuzzers run on OSS-Fuzz for a while.
    d. If bugs occur fix these.
2. We advise the Linkerd2-proxy team to monitor coverage in external dependencies as well. Fuzzing of Rust projects is still a relatively young activity, which means that a lot of projects have not been analysed yet and thus it is reasonable to expect some bugs to exist in these projects.

Conclusions and future work

In this project we integrated fuzzing into Linkerd2-proxy and several of its dependencies. During the engagement a total of 14 fuzzers were written and 8 projects were integrated into the OSS-Fuzz infrastructure (1 rejected, 4 pending, 3 accepted). A total of 2 bugs were found by the Linkerd2-proxy fuzzers, and several bugs were also found for external dependencies. The maintainers of Linkerd2-proxy as well as the maintainers of the external dependencies are aware of the bugs.

The current set up of the fuzzers is straightforward to extend on, however, it is indeed imperative to follow up on these efforts. The fuzzing set up created in this project makes for a good foundation and has shown to be fruitful. We estimate the short-term advice on fuzzing Linkerd2-proxy to be achievable within a few months by dedicating a few hours of work each week, and the long-term goals achievable within half a year. We highlight here the importance of doing this work in iterations, namely integrating new fuzzers and letting them run for a while, assess results and then repeating the process.