Spring 2021

PRESENTS

Fuzzing integration for OCI: runC and umoci

In collaboration with the Opencontainers maintainers and The Linux Foundation

Authors

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

Executive summary

Goal of engagement

The overall goal of the engagement described in this report was to integrate security and reliability analysis by way of fuzzing runC and Umoci projects. This was done in a manner such that vulnerability analysis will happen continuously (even after the engagement). RunC and Umoci are both written in Go which makes them susceptible to panics caused by issues such as index out of range, slice bounds out of range, assignment to entry in nil map and nil pointer dereferences.

Scope of engagement

The entire code base for both projects was considered. For the runC project critical dependencies were also considered.

Methodology

Ada Logic's security researchers performed an initial analysis of the two projects to determine the security criticality as well as an optimal approach to initiating a fuzzing infrastructure. Following this we started writing the actual fuzzers and creating the oss-fuzz set up.

Results summarised
17 fuzzers developed for runC and Umoci
Documentation and environment for running Umoci fuzzers locally
Integration of continuous fuzzing for runC by way of OSS-Fuzz
No bugs found, however, 9 fuzzers are pending being merged and we predict once all fuzzers have been merged that some bugs will be found.

Engagement process and methodology

Initial assessment of library

The first part of the engagement consisted of an initial assessment of both projects. This part of the engagement does not yield any fuzzers itself, however it does provide answers that are important to writing the fuzzers itself. Below we summarize these findings and what they meant for the engagement.

While the scope of the engagement included two projects, runC and Umoci, the initial assessment revealed a higher security criticality for runC than for Umoci, which meant that more time was spent on this project when writing the fuzzers.

Next, we found that runC had few targets that would be considered obvious entry points from a fuzzing perspective. When considering initial entry points for a target project, parsing routines, text processing, encoding, marshaling API's are targets that are well suited for modern fuzzing engines. There is a limited amount of this type of code in the runC codebase, and as a result we considered other ways to utilize modern fuzzing capabilities to pass random data to the project. This revealed a necessity to write a small helper library since several of the API's we wished to target take structures as input. At first we looked at the gofuzz project to create fuzzed structures, but we encountered a few shortcomings in its interface with the go-fuzz engine. This led us to write our own helper project to fuzz structures which we call go-fuzz-headers. It is inspired by gofuzz and can be found here: https://github.com/AdaLogics/go-fuzz-headers

Overview of fuzzers

Below we give an overview of the fuzzers contributed to runC and Umoci.

runC

In total 12 fuzzers were written for the runC project. The fuzzers were added in the directories of the packages they target, while a directory for utils is kept at /tests/fuzzing. Utils currently refers to the OSS-fuzz build script, and in case other utils need to be added such as seed corpora, dictionaries or general helper methods, these can be placed there.

Fuzzer name	Path	Package
FuzzCgroupReader	runc/libcontainer/cgroups/fs2	fs2
FuzzGetStats	runc/libcontainer/cgroups/fs2	fs2
Fuzz	runc/libcontainer/specconv	specconv
Fuzz	runc/libcontainer/cgroups/devices	devices
FuzzSecureJoin	runc/libcontainer/cgroups/fscommon	fscommon
FuzzFindMpDir	runc/libcontainer/intelrdt	intelrdt
FuzzSetCacheScema	runc/libcontainer/intelrdt	intelrdt
FuzzParseMonFeatures	runc/libcontainer/intelrdt	intelrdt
FuzzStateApi	runc/libcontainer	libcontainer
FuzzUser	runc/libcontainer/user	user
FuzzUIDMap	runc/libcontainer/userns	userns
FuzzUnmarshalJSON	runc/libcontainer/configs	configs

Umoci

5 fuzzers were added to the Umoci project.

Fuzzer name	Path	Package
FuzzGenerateLayer	umoci/oci/layer	layer
FuzzUnpack	umoci/oci/layer	layer
FuzzMutate	umoci/mutate	mutate
Fuzz	umoci/oci/casext	casext
Fuzz	umoci/pkg/hardening	hardening

Pull Requests with the fuzzers

In the following tables we outline the pull requests for the given fuzzers, which shows explicitly which fuzzers are still pending for analysis. During the engagement the maintainers asked the Ada Logics researchers to wait with new commits due to lack of movement in getting the fuzzers through the review process. In practice it took several weeks for most pull requests.

Runc

Fuzzer name	Link	Merged
FuzzCgroupReader	https://github.com/opencontainers/runc/pull/2879	Pending
FuzzGetStats	https://github.com/AdaLogics/runc-fuzzers/blob/main/fs2_fuzzer.go	Pending*
Fuzz	https://github.com/opencontainers/runc/pull/2864	Pending
Fuzz	https://github.com/AdaLogics/runc-fuzzers/blob/main/devices_fuzzer.go	Pending*
FuzzSecureJoin	https://github.com/opencontainers/runc/pull/2878	Pending
FuzzFindMpDir	https://github.com/AdaLogics/runc-fuzzers/blob/main/intelrdt_fuzzer.go	Pending*

FuzzSetCacheScema	https://github.com/AdaLogics/runc-fuzzers/blob/main/intelrdt_fuzzer.go	Pending*
FuzzParseMonFeatures	https://github.com/AdaLogics/runc-fuzzers/blob/main/intelrdt_fuzzer.go	Pending*
FuzzStateApi	https://github.com/opencontainers/runc/pull/2848	Pending
FuzzUser	https://github.com/opencontainers/runc/pull/2841	Merged
FuzzUIDMap	https://github.com/opencontainers/runc/pull/2841	Merged
FuzzUnmarshalJSON	https://github.com/opencontainers/runc/pull/2841	Merged

*Upon request by maintainers, not committed:
https://github.com/opencontainers/runc/pull/2878#issuecomment-812203536

Umoci

Fuzzer name	Path	Package
FuzzGenerateLayer	https://github.com/opencontainers/umoci/pull/365	Merged
FuzzUnpack	https://github.com/opencontainers/umoci/pull/371	Merged
FuzzMutate	https://github.com/opencontainers/umoci/pull/371	Merged
Fuzz	https://github.com/opencontainers/umoci/pull/362	Merged
Fuzz	https://github.com/opencontainers/umoci/pull/362	Merged

Results

The fuzzers found no bug during the assessment, which is a great achievement to the RunC and Umoci authors. However, we acknowledge that there is a reasonable expectation that bugs will occur once the pending pull requests are merged in. We go into details with this in the next section.

Advice following engagement

In this section we outline the recommended next steps for fuzzing runC and umoci. We divide the recommendations into short-term action steps followed by long-term, strategic steps to further develop the projects' fuzzing capabilities.

Short-term advice

1: Merge the pending fuzzers

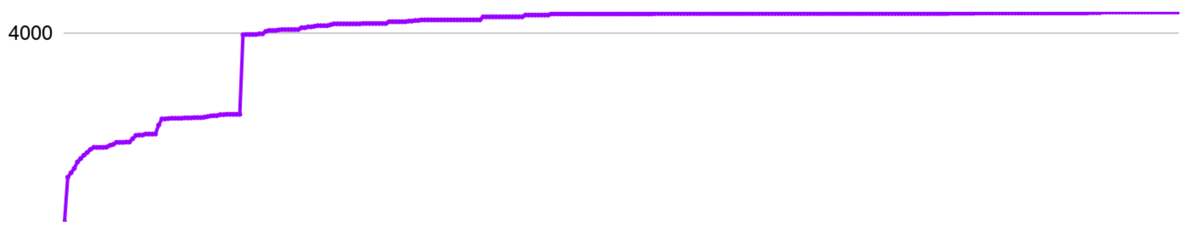
The primary advice in the short-term is to merge in any of the pending fuzzers. During the engagement, there was not enough movement on the PRs and a fair amount of the fuzzers are still pending. These fuzzers are valuable and should be top priority in terms of moving fuzzing RunC and Umoci forward.
We recommended moving the currently unmerged fuzzers for runC to either runC's own repository or to a repository dedicated to hosting fuzzers for the project. Storing the fuzzers on runC's main repository will allow for contributors to quickly get a sense of what has already been done in terms of fuzzing the project.

The fuzzers make use of Ada Logics' go-fuzz-headers library which introduces a new dependency to runC. This dependency is not used in production and only by the fuzzers, and since runC uses vendoring, the runC maintainers may want to consider alternative integration strategies. In doing so, it is Ada Logics' recommendation to not postpone running the fuzzers entirely, but instead start running the fuzzers while establishing a holistic solution to adding the dependency. In case this is an issue that runC's maintainers wish to resolve, Ada Logics offer the following suggestion:

First of all, start running the fuzzers from Ada Logics' runc-fuzzers repository. Next, move the fuzzers upstream without including the go-fuzz-headers vendor, the following actionable steps can be taken:

- Merge the fuzzers into runC's main repository.
- Exclude fuzzers from being checked during CI tests to eliminate build errors from missing dependencies in the vendor folder.
- Install the fuzzing-dependencies when building them in OSS-fuzz like is done here.

We want to stress the importance of continuous fuzzing as the fuzzers for Umoci and runC continue to grow in coverage over time. As an example, consider the following figure which displays data from an experiment of running the StateApiFuzzer for 6 hours. The x-axis plots time and the y-axis plots coverage:



Thus, running all fuzzers continuously will allow them to explore coverage in their theoretical reach. Even after all code that a fuzzer can explore has been explored, it is recommended to keep running them continuously. Changes in upstream code may allow fuzzers to reach new parts of the project, and even so, there are examples of bugs having been found by fuzzers several cpu years after coverage stopped increasing.

RunC can benefit from its integration into OSS-fuzz to run the fuzzers continuously. All fuzzers added to the upstream build script run through regular scheduled jobs and the maintainers get notified whenever bugs are found. Additionally, OSS-fuzz builds up the corpus over time which offers a fairly hands-off solution for the runC maintainers.

To help maintainers and contributors of Umoci execute the fuzzers continuously we added documentation on running them locally. All that is required to do that is a local installation of

Docker. Naturally, we encourage everyone to run the fuzzers locally and report any issues responsibly.

Link to repository containing runC's unmerged fuzzers:
https://github.com/AdaLogics/runc-fuzzers
Link to pull request to build runC's unmerged fuzzers:
https://github.com/opencontainers/runc/pull/2914

Once the above has been achieved, it is recommended to allow the fuzzers to run continuously for a couple of weeks before any further modifications are made. This is to verify that the fuzzers can indeed run continuously without interruptions such as crashes that do not qualify as valid bugs. Such interruptions could arise from edge cases in the input generated by the fuzzing engine that breaks the fuzzer or causes it to time out or run out of memory. Once the fuzzers have been verified to run continuously, a few low-effort steps can be taken to increase the value by the fuzzers. We list these below.

Long-term advice

In this section we outline two pieces of advice for the long-term.

1: Fuzzing during CI tests

The fuzzers can be run during CI tests for a few minutes to catch any immediate bugs introduced in the code committed. This will help discover bugs before being merged into the project as well as support the continuous fuzzing efforts since the fuzzers will not be blocked by bugs that are found within a short fuzz run. This will allow the fuzzers to run continuously by OSS-fuzz to test for harder-to-find bugs. runC qualifies for CIFuzz as it is integrated into OSS-fuzz.

2: runC: Add support for running processes in containers

During the engagement, Ada Logics looked at the possibilities for running processes in containers created by a fuzzer in the runC project. Given the current state of fuzzing maturity of the project, it was deemed out of scope to enable this during an initial fuzzing infrastructure set up.

For future reference, the steps in /libcontainer/README.md can be followed until the point where a process is run:

```
err := container.Run(process)
if err != nil {
	container.Destroy()
	logrus.Fatal(err)
	return
}
```

While the process at this point does exit gracefully, it is not executed successfully. Adding support for processes to be run by containers created by fuzzers will allow contributors to write fuzzers that test for critical bugs like container escapes.
The fuzzer Ada Logics used to attempt executing processes against containers inside fuzzers can be sent to the runC maintainers upon request.

Conclusions

As part of this engagement, 17 fuzzers have been developed for runC and Umoci that target parts of the codebase in which there have previously been bugs. No bugs were found during the engagement. However, we note here that a significant portion of the fuzzers are still pending merge and thus pending being run for a longer period of time. This means a significant part of the code base is still to be analysed and we can only get a complete understanding of the project in terms of bugs once the PRs get merged. The fuzzing suite integrated is highly autonomous to a degree where the fuzzers of runC require less effort to run than the unit test suite and the fuzzing suite provided for Umoci requires a few steps to execute locally.