HOCHSCHULE MÜNCHEN UNIVERSITY OF APPLIED SCIENCES

Exposé for Master of Science's Thesis in Informatics

Cyber risk reduction through transparency for Zig packages

David Pierre Sugar

Contents

1 Expose

Vulnerabilities in software components often have an profound impact on end users and companies alike. While secure coding practices and vulnerability analyses help mitigate the introduction of new vulnerabilities and aid in the detection of existing ones, it is almost impossible to prevent vulnerabilities in general. For this reason, it is important to have procedures and tooling in place that allow to efficiently act on the discovery of vulnerabilities.

1.1 Motivation

Zig is a comparatively young programming language, missing tooling in many areas including security. To our knowledge there are only few tools available that help with adding software transparency and auditing capabilities to Zig packages. Without such tooling it is hard for developers to generate and communicate important, security-related information about their software and for users to make informed decisions on what software to use, update or fix.

1.2 Research question

In the proposed thesis we want to design and develop the required tooling for adding software transparency and auditing capabilities to Zig packages. Furthermore, we plan to make the tooling openly available at an early stage to investigate how such tooling is adopted by the Zig community and gather feedback that can be transferred back into the design and development process. The tools developed as part of the proposed thesis are evaluated using real-world packages.

1.3 Related Work

There exists a multitude of vulnerability databases with the Rust Security Advisory Database[1] and the Open Source Vulnerabilities (OSV) database[2] being two notable ones. Also, those databases share disclosed vulnerabilities to a certain extend, e.g., the Rust Advisory Database also publishes advisories to OSV. In general, there is a notable overlap between the work of the Rust Secure Code Working Group[3] and the goals stated in this Expose. This includes cargo-audit[4] a tool for auditing Rust projects for vulnerable

[1]https://rustsec.org/

[2]https://google.github.io/osv.dev/

[3]https://www.rust-lang.org/governance/wgs/wg-secure-code

[4]https://github.com/rustsec/rustsec

dependencies sourced from the Rust Security Advisory Database and cargo-supply-chain[5] a tool for gathering information about Rust crates. The CycloneDX organization on Github has also published a module for Cargo[6] and other languages[7] (not including Zig). There exist also tools like action-owasp-dependency-track-check[8] which generate BOMs for projects and then perform vulnerability checks using those BOMs. Online services like Vulert[9] are capable of detecting vulnerabilities for software based on SBOMs.

To our knowledge, none of the existing tools support the Zig programming language.

1.4 Approach and methodology

The goal of this thesis is to develop and evaluate tooling and guidelines for securing software components in the Zig ecosystem. This includes but is not limited to the publication of advisories, i.e. the description of vulnerabilities and issues found in Zig packages, as well as the auditing of existing packages based on dependencies. Therefore, we make the following open source contributions to the Zig ecosystem.

First, we develop a platform for submitting, reviewing and publishing advisories based on Git, with Github being to platform of choice, similar to the Rust Security Advisory Database. This platform allows the submission of advisories for Zig packages via pull-requests. Each advisory is defined using the Zig Object Notation (ZON). All published advisories get assigned a unique identifier and are grouped based on the fingerprint of the corresponding package. The ZON-advisories can later be translated into other formats like a CycloneDX ("Authoritive Guide to Sbom" 2024) Vulnerability Disclosure Report[10] and imported by other databases.

Second, we develop a command line tool called the Zig Audit Tool (ZAT) for helping with the audit of Zig packages. The tool will analyze a packages build.zig.zon and generate a CycloneDX SBOM for it, incorporating all listed dependencies. During this process, ZAT will use the advisories published to the Zig Advisory Database[11] to find vulnerabilities that are applicable to the package (including transitive dependencies) and generate appropriate warnings or errors[12]. VDRs are generated to communicate

[5] https://github.com/rust-secure-code/cargo-supply-chain?tab=readme-ov-file
[6] https://github.com/CycloneDX/cyclonedx-rust-cargo
[7] https://github.com/orgs/CycloneDX/repositories
[8] https://github.com/Quobis/action-owasp-dependecy-track-check
[9] https://vulert.com/abom
[10] https://cyclonedx.org/use-cases/vulnerability-disclosure/
[11] At a later stage maybe also other databases.
[12] How those warnings or errors look like will be investigated as part of the thesis.

issues with a package to potential users and tools that read the BOM[13].

A secondary goal is to make ZAT also available via Github Actions to fully automate the auditing and generation of SBOMs for Zig packages, e.g., for rejecting a push if a vulnerability has been found in one of the dependencies. Of course, one must also be cable to override such rejections, e.g., if the vulnerability is not exploitable for a given configuration.

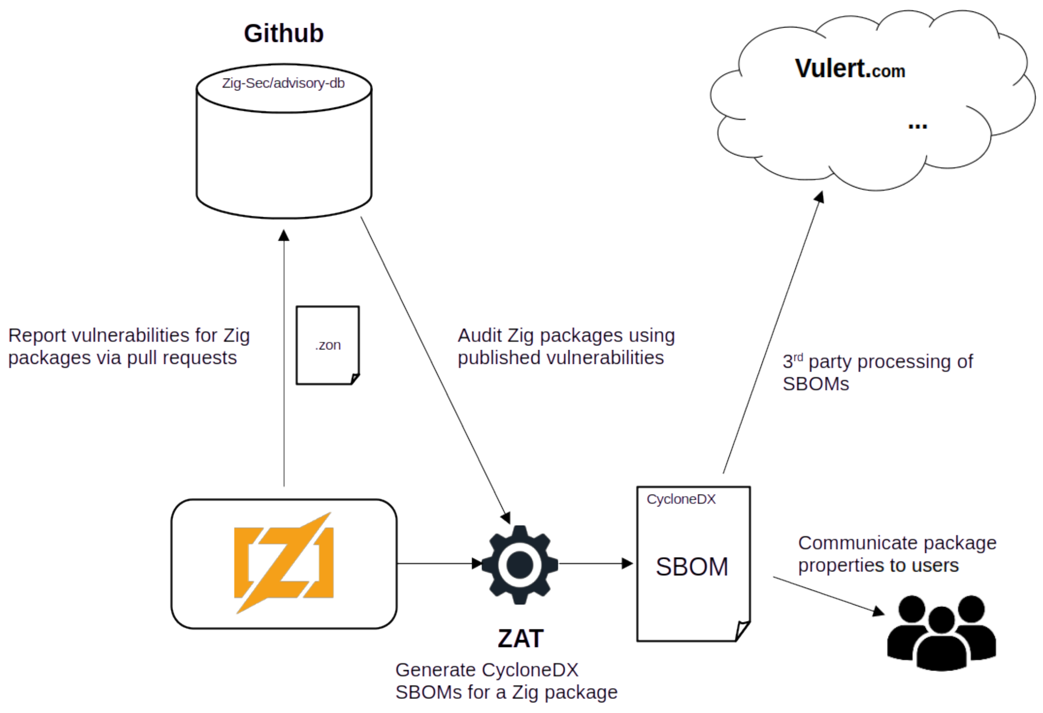
Github
Zig-Sec/advisory-db
Vulert.com
...
Report vulnerabilities for Zig packages via pull requests
.zon
Audit Zig packages using published vulnerabilities
3rd party processing of SBOMs
CycloneDX
SBOM
Communicate package properties to users
ZAT
Generate CycloneDX SBOMs for a Zig package

Figure 1: A sketch of how all proposed tools work together.

One challenging problem is to reliably find advisories for libraries that are not directly listed in a build.zig.zon but are specified within a build.zig as a linked library. Tackling this issue is also a secondary goal and could be part of future research.

The developed tools will be evaluated in two ways. First, through manual tests executed by the author using existing repositories. For those tests a set of interconnected repositories will be selected, with a dependency depth of at least 3 levels. For this set, audits will be executed using ZAT after filing advisories for different packages of the given set. The results will be evaluated manually. One project that fits the stated criteria is PassKeeZ, a passkey implementation for Linux. Second, through users of the community that report vulnerabilities and use the ZAT tool for their own packages. To

[13]How those VDRs look like and how they fit into the bigger picture is part of the research.

what extend the second evaluation approach will succeed is up to the Zig community, i.e. it is just an addition to the first evaluation approach.

Appendix A: Supplementary Material

– Supplementary Material –

✏️ **Write here the appendix**

List of Acronyms

List of Figures

List of Tables

Bibliography

[1] “Authoritive Guide to SBOM.” [Online]. Available: https://cyclonedx.org/guides/OWASP_CycloneDX-Authoritative-Guide-to-SBOM-en.pdf