




CoPhi - Mining C/C++ Packages for Conan Ecosystem Analysis

Vivek Sarkar 
University of Washington
Seattle, USA
viveksar@cs.washington.edu

Anemone Kampkötter 
Department of Computer Science
Technische Universität Dortmund
Dortmund, Germany
anemone.kampkoetter@tu-dortmund.de

Ben Hermann 
Department of Computer Science
Technische Universität Dortmund
Dortmund, Germany
ben.hermann@cs.tu-dortmund.de

Abstract—Large-scale analyses of software ecosystems allow researchers to identify widespread vulnerabilities, validate dependencies for safe usage, and gain an understanding of the conditions of software package landscapes. In the C/C++ ecosystem however, there are many challenges facing large-scale analyses, due to the lack of a standard package manager or build system. With this work, we aim to explore the Conan ecosystem by statically analyzing it as a whole and on a large scale. We provide a static analysis tool set named CoPhi that crawls Conan packages and analyzes them for specific features to capture C/C++ ecosystem metrics of interest, and also create corpora with user-defined properties. In a case study, we demonstrate the effectiveness of CoPhi by analyzing 620 Conan packages for four different metrics.

Index Terms—Mining Software Repositories, C/C++, Static Analysis, Conan Package Manager

I. INTRODUCTION

Software reuse is a common practice in software engineering that is increasingly administered by sophisticated package managers. For the C/C++ ecosystem, Conan [1] has become one of the most popular open-source repositories among application-level package managers. While it is recommended to use unifying package management tools, these are still not yet widely used [2]. Package managers enable structured dependency management and thus facilitate the reuse of software and the analysis of third-party libraries (TPLs) or dependencies, especially on a large scale. The lack of a standardized package manager for the entire software lifecycle, which also includes the TPL dependencies, is associated with a high level of manual effort and facilitates the infiltration of security vulnerabilities through external and possibly inconsistent libraries.

Therefore, with the reuse of software and the increasing use of open-source libraries, it becomes more and more relevant to validate the quality of these third-party components. Many standalone tools have been developed for the C/C++ ecosystem with the goal of vulnerability identification to address this issue, but such external tools have their limitations and can be cumbersome to use [3], [4]. To advance entire ecosystem analysis, Buchkova et al. [5] build a special dataset, for which they mine metadata from package managers including Conan. With this work, they can acquire metadata-related metrics by collecting packages, versions, and dependencies. Such mining

of entire ecosystems facilitates the creation of datasets required for large-scale static analyses.

In our work, we propose a static code analysis of unified TPLs in Conan to explore the C/C++ ecosystem in an automated way and on a large scale. We build a static analysis tool set called CoPhi that crawls all Conan packages from Conan’s public repository ConanCenter and collects features of the binaries provided by each package to capture specific metrics. The crawled binaries are sequentially analyzed using the PhASAR C/C++ analysis framework [6], which is capable of solving sophisticated data-flow problems as well as a plethora of other static analysis objectives. In a case study, we perform extensible analyses on 620 Conan packages. We capture the following metrics: the number of binaries in a package that are executables as opposed to libraries, the call-graph sizes of each binary in a package, the LLVM instruction numbers contained in each package, and the inheritance depths of the Conan packages under analysis.

CoPhi allows for a targeted search for software artifacts that can further facilitate the execution of experiments and analyses on a large scale. It enables to identify packages containing specific software items and use them to generate special-purpose benchmarks, inspired by the work of Reif et al. [7] for the Java ecosystem. In particular, research and evaluation of (new) C/C++ analyses with PhASAR can be carried out efficiently on a large scale. To our knowledge, there has been no such code-based Conan ecosystem analysis to date.

Our work makes the following contributions:

- We develop a metrics and static analysis pipeline for C/C++-based software artifacts mined from Conan packages using PhASAR. This allows us to automatically analyze a wide range of C/C++ packages.
- To exemplify our tool, we conduct a case study in which we examine four different metrics in the C/C++ ecosystem and show that CoPhi is effective and flexible in analyzing queries of interest.
- We offer an easy and built-in solution to perform large-scale studies on representative C/C++ software packages with a variety of static analysis options.

II. COPHI ARCHITECTURE

CoPhi consists of three modularized components, which are

containerized for cross-platform usage. The first is the scraper, which traverses ConanCenter and extracts a set of packages. Each package is made up of a set of LLVM Modules as well as additional metadata provided by Conan. The second tool is the extractor, which runs a set of queries on the packages using the static analysis framework PhASAR. This creates a map from packages to sets of features on those packages. The third tool is the filterer, which takes the feature map constructed by the extractor, and given a set of desired features, outputs the packages matching those features. We describe these components with more detail in the next three sections.

A. ConanCenter Scraper

ConanCenter [8] is the main public repository on which Conan projects are hosted, making it an ideal target from which to create large corpora. Conan has a concept of recipes, which are build scripts for Conan packages. When built, these packages contain one or more binaries (both executables and libraries), as well as some metadata about the built package itself. The Conan Center Index [9] contains all the recipes for the packages hosted on ConanCenter. Our scraper has a copy of this repository, which it uses to build the packages. We take as input the number of packages to scrape, and then choose that many packages randomly from the repository. We then modify the compilation of packages in two ways to make the results usable by PhASAR. Firstly, as PhASAR works on LLVM bitcode Modules, and not native binaries or C/C++ source code, we need to produce LLVM Modules during the build process. To do this, we utilize `gllvm` [10], which is a wrapper around `clang` that saves the LLVM bitcode produced during build time, and from them, constructs the LLVM bitcode module for the corresponding binary. Secondly, we build all the binaries of a package in debug mode using a Conan option, as this provides PhASAR with more information on the original source code. In total then, the information that we store for each package consists of the LLVM Modules corresponding to each of its binaries, along with some package metadata we additionally scrape.

B. Feature Extractor

The extractor receives as input a set of packages, and outputs a map from packages to sets of features characterizing them. These features are produced by static analyses, a set of which is also given as input to the extractor, along with the order in which to run the given queries on a package.

1) *Queries and Features*: Queries are static analyses, which given a package, use PhASAR to analyze it and then output a set of features attached to that package. Queries can derive one or more unique features, and assign zero or more of those features to a specific package when analyzing it. Queries are easily extendable, only requiring one to extend a single interface, and implement a single method, which returns the set of features. Listing 1 exemplifies the implementation of a query that computes the number of instructions for a given package.

```
bool NumInstructionsQuery::runOn(
Package const * const pkg,
Query::Result * const res,
const shared_ptr<atomic_bool> &terminate) const
{
    const FeatureID fid(
        *static_cast<Query const *>(this),
        Type::UNIT, Attribute::Type::U_INT,
        FeatureData::Type::BINMAP);
    BinAttrMap num_instrs_map(Attribute::Type::U_INT);
    for (const auto &bin : pkg->bins()) {
        if (*terminate)
            return false;
        size_t n_instrs = 0;
        for (const auto &F : bin->getModuleRef())
            for (const auto &BB : F)
                n_instrs += distance(BB.begin(), BB.end());
        num_instrs_map.insert(
            bin->getID(),
            Attribute(n_instrs));
    }
    res->emplace(fid, FeatureData(num_instrs_map));
    return true;
}
```

Listing 1: Example of a query implementation.

A feature itself is uniquely identified by its name and which query produced it. In addition to representing the existence of some property, features can also carry data inside of themselves. Currently, these data can be chosen from a handful of primitive types. However, the user can choose, instead of storing data on the package as a whole, to store said data instead on the individual binaries. This, combined with the fact that queries can derive multiple features, enables queries to be very flexible in how they represent the results of their static analysis. We provide an example of a query and the features it derives in Figure 1. Note that different features derived from the same query can hold different data (or none at all).

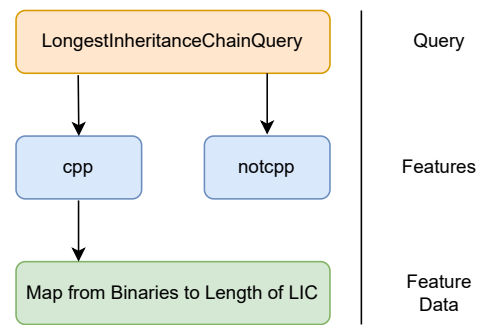


Fig. 1: Example of a query, the features it derives, and the data that those features hold.

2) *Timeouts*: Due to memory constraints, we also introduce a way to time out on packages which take too long to be analyzed or which have too many binaries. As Conan packages can contain any number of binaries, reifying a single package could potentially be very costly, which is why we allow the

user to set a limit on the number of binaries a package may have. We also allow the user to set a time limit for running all given queries on a single package, in order to abort potentially intractable analyses on a given hardware platform. If a package exceeds the maximum number of binaries, it is skipped over. If it exceeds the time limit for evaluation, the analysis is suspended.

C. Filterer

The filterer, the final tool, takes in the feature map created by the extractor along with a list of filters and produces a smaller feature map consisting of the packages which pass all the filters. Filters are, firstly, identified by the feature which they look for. Secondly and optionally for features with data, the filter can be accompanied by an additional restriction on the data the feature carries. For example, if the feature carries a boolean or a string along with it, the user can specify the value to which it should be equal, and for integers or doubles, the user can specify a range within which the value should fall. For features which carry data on each of their binaries, it can be further specified whether all binaries have to pass the additional restriction or only one. An example of a filter definition can be found in the following Listing 2.

```
{
  "feature_id": {
    "name": "LongestInheritanceChainQuery",
    "type": "cpp",
    "attr_type": "uint",
    "data_type": "binmap"
  },
  "use_range": true,
  "filter_type": "exists",
  "range": {
    "attr_type": "uint",
    "lower_bound": 2,
    "upper_bound": 5
  }
}
```

Listing 2: Example filter that accepts packages with at least one binary whose longest inheritance chain is in the range [2, 5).

III. TOOL USAGE

CoPhi mines and statically analyzes Conan packages and enables the creation of corpora of C/C++ packages with specific desired features. The tool is divided into three Docker containers: the scraper, the feature extractor, and the filterer, which make it easy to use. The corresponding images can be built from the source files in our provided artifact [11]. New queries can be implemented by extending the `Query` interface and registering it accordingly. A detailed installation description and further information including required command-line parameters can also be found in our artifact. We also provide the code for the queries used in our case study below and make the data available for reuse.

IV. CASE STUDY

To test CoPhi we randomly scraped 700 projects from ConanCenter, from around 1600 projects in total, and ran four queries on them, each of which is referred to in its own subsection below. The timeout to evaluate a package was set at 15 minutes and the maximum number of binaries we allowed in a package was 20. With these settings, the feature extractor was able to analyze 620 out of the total of 700 packages over a period of 33 hours. The analyses of the remaining 80 packages were not successfully completed.

A. Executable vs. Library

This query determines whether each binary in a package is either an executable or a library, and took an average of 0.0033 seconds to run on one package. We determined whether a binary was an executable or not by looking for the presence of a main method in the LLVM Module. The following Figure 2 illustrates the relative amounts of executables vs. libraries in the packages from ConanCenter. Most packages consist of one executable file and two libraries.

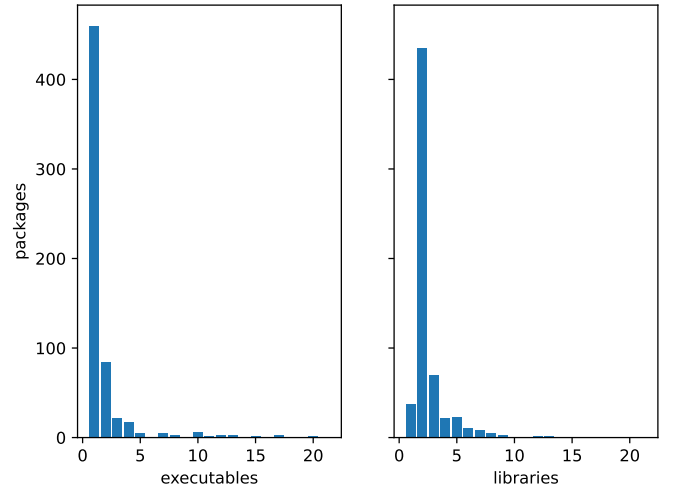


Fig. 2: Number of executables vs. number of libraries per package.

B. Instruction Number

This query finds the total number of LLVM instructions in each binary in a package, and took an average of 0.0358 seconds to run on a package. Figure 3 shows the distribution of instruction numbers per binary. We can see that most binaries include a total of between 10^4 and $5 \cdot 10^5$ instructions.

C. Inheritance Depth

This query measures the lengths of the longest (virtual) inheritance chains in a package's binaries. However, as this metric is only applicable to binaries derived from C++ source code, and not C, we must first determine whether a package is written in C++. We do this conservatively, by analyzing

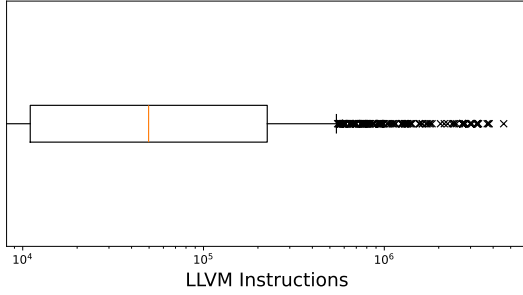


Fig. 3: The distribution of LLVM instructions per binary.

packages’ metadata. Of the 620 packages, we found 287 to be written in C++. We assigned those packages a feature with attached data indicating the longest lengths for each binary in the package. The query took an average of 2.69 seconds to run on a single package. From the distribution seen in Figure 4, we see that the median chain length is 1. This low number is most likely due to PhASAR’s restriction of only counting virtual inheritance.

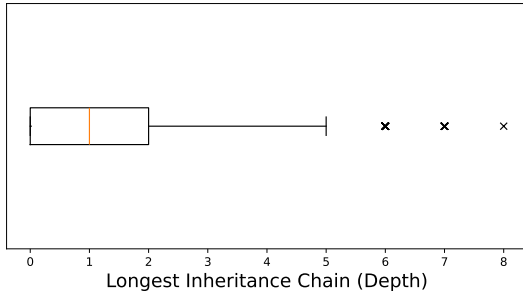


Fig. 4: The distribution of the lengths of the longest inheritance chains per binary.

D. Call-Graph Size

This query determines the call graph size for each binary in a package - namely, the number of edges and nodes. It took an average of 48.38 seconds to run on a single package. Due to the timeout limit, only 533 out of the 620 were able to be successfully analyzed. When generating the call graph, we change the entrypoint functions depending on the type of binary. For executables, we set the main function as the entrypoint, while in libraries, we set all functions in the LLVM Module as entrypoints. Figure 5 illustrates the number of call edges on the y-axis and the amount of nodes in the call graphs on the x-axis.

V. LIMITATIONS

The main limitation of the CoPhi tool is its reliance on Conan, which obviously affects the number of packages able to be analyzed. Packages not represented, such as system libraries not present on ConanCenter, simply cannot be included in

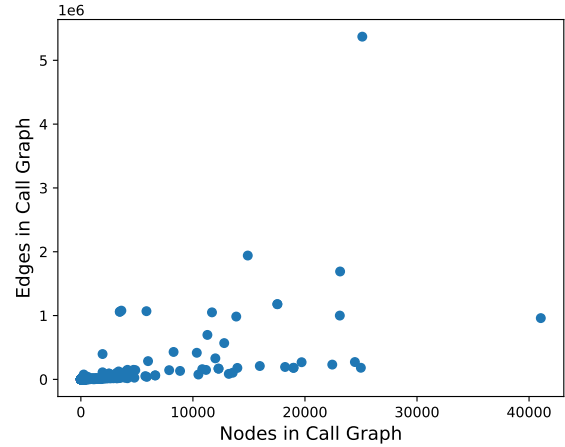


Fig. 5: Call graph nodes vs. edges per binary.

any corpora. Further, despite the convenience containerization affords us, it also prevents us from compiling any packages not compatible with the container environment (e.g., packages specifically designed for only Windows).

VI. CONCLUSION

In this work, we presented CoPhi, a tool set that mines and statically analyzes C/C++ packages in the Conan ecosystem. CoPhi’s scraper crawls Conan packages, which are then analyzed by the C/C++ analysis framework PhASAR, and finally filtered to summarize the packages containing the analyzed metric. Our case study shows that CoPhi is an effective and flexible tool to conduct Conan ecosystem analysis or to collect a number of C/C++ packages with specific properties for corpora creation. CoPhi can easily be extended by custom queries, allowing others to apply their analyses at large scale.

VII. FUTURE REUSABILITY

Possible future uses for this tool include large-scale empirical studies, for example dependency and vulnerability analyses. Additionally, the development of new static analyses could be supported through the creation of corpora used to test said analyses.

DATA AVAILABILITY

Our artifact [11] contains the data for our case study, the source code for CoPhi, and some example configuration and filtering files. The file `cophi_instructions.pdf` provides detailed instructions on how to make and run the CoPhi containers. We additionally provide a sample dataset of 15 packages already scraped.

ACKNOWLEDGEMENTS

Vivek Sarkar worked on the research presented here during his internship at TU Dortmund in Summer 2024 funded by the German Academic Exchange Service (DAAD) in the RISE Germany funding program. The authors would like to thank the DAAD for enabling the collaboration which made this project possible.

REFERENCES

- [1] “Conan - C/C++ Package Manager,” <https://conan.io/>, accessed 2024-11-07.
- [2] W. Tang, Z. Xu, C. Liu, J. Wu, S. Yang, Y. Li, P. Luo, and Y. Liu, “Towards understanding third-party library dependency in c/c++ ecosystem,” in *Proceedings of the 37th IEEE/ACM International Conference on Automated Software Engineering*, ser. ASE ’22. New York, NY, USA: Association for Computing Machinery, 2023. [Online]. Available: <https://doi.org/10.1145/3551349.3560432>
- [3] S. Li, Y. Wang, C. Dong, S. Yang, H. Li, H. Sun, Z. Lang, Z. Chen, W. Wang, H. Zhu, and L. Sun, “Libam: An area matching framework for detecting third-party libraries in binaries,” *ACM Trans. Softw. Eng. Methodol.*, vol. 33, no. 2, Dec. 2023. [Online]. Available: <https://doi.org/10.1145/3625294>
- [4] J. Wu, Z. Xu, W. Tang, L. Zhang, Y. Wu, C. Liu, K. Sun, L. Zhao, and Y. Liu, “Ossfp: Precise and scalable c/c++ third-party library detection using fingerprinting functions,” in *2023 IEEE/ACM 45th International Conference on Software Engineering (ICSE)*, 2023, pp. 270–282.
- [5] P. Buchkova, J. H. Hinnerskov, K. Olsen, and R.-H. Pfeiffer, “Dasea: a dataset for software ecosystem analysis,” in *Proceedings of the 19th International Conference on Mining Software Repositories*, ser. MSR ’22. New York, NY, USA: Association for Computing Machinery, 2022, p. 388–392. [Online]. Available: <https://doi.org/10.1145/3524842.3528004>
- [6] P. D. Schubert, B. Hermann, and E. Bodden, “Phasar: An inter-procedural static analysis framework for c/c++,” in *Tools and Algorithms for the Construction and Analysis of Systems*, T. Vojnar and L. Zhang, Eds. Cham: Springer International Publishing, 2019, pp. 393–410.
- [7] M. Reif, M. Eichberg, B. Hermann, and M. Mezini, “Hermes: assessment and creation of effective test corpora,” in *Proceedings of the 6th ACM SIGPLAN International Workshop on State Of the Art in Program Analysis*, ser. SOAP 2017. New York, NY, USA: Association for Computing Machinery, 2017, p. 43–48. [Online]. Available: <https://doi.org/10.1145/3088515.3088523>
- [8] “The Conan libraries and tools central repository,” <https://conan.io/center>, accessed 2024-11-25.
- [9] “ConanCenter Index,” <https://github.com/conan-io/conan-center-index>, accessed 2024-11-25.
- [10] “GLLVM,” <https://github.com/SRI-CSL/gllvm>, accessed 2024-11-25.
- [11] V. Sarkar, A. Kampkötter, and B. Hermann, “CoPhi Artifact,” <https://doi.org/10.5281/zenodo.14226786>, November 2024.