PurlDB

AboutCode.org authors and contributors

May 10, 2024

CONTENTS

CHAPTER

ONE

GETTING STARTED

1.1 Getting Started

1.1.1 Installation

Instruction for a standalone purldb installation.

1.1.2 Installation with ScanCode.io

Installation documentation to set up purldb and SCIO instances together so we have:

- a main SCIO server for scanning and matching code
- a purldb instance for storing package scans and matching data
- SCIO workers for the purldb instance to scan and index packages.

1.1.3 Contirbute

Documentation to support code and documentation contributions to purldb.

CHAPTER

TWO

PURLDB

PURLdb is a database of packages, with package metadata and indexes for package files and archives, and various API endpoints to get data about these packages and match to other codebases.

2.1 PURLdb

2.1.1 The purldb

This repo consists of four main tools:

- PackageDB that is the reference model (based on ScanCode toolkit) that contains package data with purl (Package URLs) being a first class citizen.
- MineCode that contains utilities to mine package repositories
- MatchCode that contains utilities to index package metadata and resources for matching
- MatchCode.io that provides package matching functionalities for codebases
- ClearCode that contains utilities to mine Clearlydefined for package data

These are designed to be used first for reference such that one can query for packages by purl and validate purl existence.

In the future, the collected packages will be used as reference for dependency resolution, as a reference knowledge base for all package data, as a reference for vulnerable range resolution and more.

Installation

Requirements

- Debian-based Linux distribution
- Python 3.11 or later
- Postgres 13
- git
- scancode-toolkit runtime dependencies (https://scancode-toolkit.readthedocs.io/en/stable/getting-started/install.html#install-prerequisites)
- `libpq-dev`
- If you are using Debian or Ubuntu : `sudo apt install libpq-dev`
- If you are using Fedora: `sudo dnf install libpq-devel`

Once the prerequisites have been installed, set up PurlDB with the following commands:

```
git clone https://github.com/nexb/purldb
cd purldb
make dev
make envfile
make postgres
make postgres_matchcodeio
```

Indexing some PURLs requires a GitHub API token. Please add your GitHub API key to the *.env* file

```
GH_TOKEN=your-github-api
```

Once PurlDB and the database has been set up, run tests to ensure functionality:

```
make test
```

Usage

Start the PurlDB server by running:

```
make run
```

Start the MatchCode.io server by running:

```
make run_matchcodeio
```

To start visiting upstream package repositories for package metadata:

```
make run_visit
```

To populate the PackageDB using visited package metadata:

```
make run_map
```

Populating Package Resource Data

The Resources of Packages can be collected using the scan queue. By default, a scan request will be created for each mapped Package.

Given that you have access to a ScanCode.io instance, the following environment variables will have to be set for the scan queue commands to work:

```
SCANCODEIO_URL=<ScanCode.io API URL>
SCANCODEIO_API_KEY=<ScanCode.io API Key>
```

Package Resource data can also be gathered by running ClearCode, where Package scan data from clearlydefined is collected and its results are used to create Packages and Resources.

```
make clearsync
```

After some ClearlyDefined harvests and definitions have been obtained, run `clearindex` to create Packages and Resources from the harvests and definitions.

```
make clearindex
```

The Package and Package Resource information will be used to create the matching indices.

Once the PackageDB has been populated, run the following command to create the matching indices from the collected Package data:

```
make index_packages
```

PurlDB API Endpoints

- `api/packages`
  - Contains all of the Packages stored in the PackageDB
- `api/resources`
  - Contains all of the Resources stored in the PackageDB
- `api/cditems`
  - Contains the visited ClearlyDefined harvests or definitions
- `api/approximate_directory_content_index`
  - Contains the directory content fingerprints for Packages with Resources
  - Used to check if a directory and the files under it is from a known Package using the SHA1 values of the files
- `api/approximate_directory_structure_index`
  - Contains the directory structure fingerprints for Packages with Resources
  - Used to check if a directory and the files under it is from a known Package using the name of the files
- `api/exact_file_index`
  - Contains the SHA1 values of Package Resources
  - Used to check the SHA1 values of files from a scan to see what Packages also has that file
- `api/exact_package_archive_index`
  - Contains the SHA1 values of Package archives
  - Used to check the SHA1 values of archives from a scan to determine if they are known Packages

MatchCode.io

MatchCode.io is a Django app, based off of ScanCode.io, that exposes one API endpoint, `api/matching`, which takes a ScanCode.io codebase scan, and performs Package matching on it.

Currently, it performs three matching steps:

- Match codebase resources against the Packages in the PackageDB
- Match codebase resources against the Resources in the PackageDB
- Match codebase directories against the directory matching indices of MatchCode

MatchCode.io API Endpoints

- `api/matching`
  - Performs Package matching on an uploaded ScanCode.io scan
  - Intended to be used with the `match_to_purldb` pipeline in ScanCode.io

Docker Setup for Local Development and Testing

PurlDB and MatchCode.io are two separate Django apps. In order to run both of these Django apps on the same host, we need to use Traefik.

Traefik is an edge router that receives requests and finds out which services are responsible for handling them. In the docker-compose.yml files for PurlDB and MatchCode.io, we have made these two services part of the same Docker network and set up the routes for each service.

All requests to the host go to the PurlDB service, but requests that go to the `api/matching` endpoint are routed to the MatchCode.io service.

To run PurlDB and Matchcode.io with Docker:

```
docker compose -f docker-compose.yml up -d
docker compose -f docker-compose.matchcodeio.yml up -d
```

Funding

This project was funded through the NGI Assure Fund https://nlnet.nl/assure, a fund established by NLnet https://nlnet.nl/ with financial support from the European Commission's Next Generation Internet programme, under the aegis of DG Communications Networks, Content and Technology under grant agreement No 957073.

This project is also funded through grants from the Google Summer of Code program, continuing support and sponsoring from nexB Inc. and generous donations from multiple sponsors.

License

2.1.2 Symbol and String Collection

The package indexing endpoint now also supports the symbol and string collection pipeline and stores them in the `extra_data` field of the resource.

How it works

When PurlDB receives an index request for a PURL via the `/api/collect` endpoint along with the symbol/string addon_pipeline, it fetches the archive download_url and creates a package for the PURL with relevant metadata. Thereafter, a scan job is scheduled which downloads the archive of the PURL and runs the scan_single_package package pipeline. Scan job also runs the requested addon_pipelines. Upon completion of the scan job, the package is updated with resource data along with the `source_symbols` and `source_strings` in the `extra_data` field of resources.

Currently PurlDB supports these addon_pipeline for symbol/string collection.

- `collect_symbols_ctags`
- `collect_strings_gettext`
- `collect_symbols_tree_sitter`
- `collect_symbols_pygments`

See the detailed tutorial on *How To get symbols and strings from a PURL/package* in PurlDB.

To use these pipeline on ScanCode.io refer to Symbol and String Collection.
For more details on these plugins refer to source-inspector.

CHAPTER THREE

PURLDB TOOLKIT

purldb-toolkit is command line utility and library to use the PurlDB, its API and various related libraries.

3.1 purldb-toolkit

purldb-toolkit is command line utility and library to use the PurlDB, its API and various related libraries.

The `purlcli` command acts as a client to the PurlDB REST API end point(s) to expose PURL services. It serves both as a tool, as a library and as an example on how to use the services programmatically.

3.1.1 Installation

pip install purldb-toolkit

3.1.2 Usage

Use this command to get basic help:

```
$ purlcli --help
Usage: purlcli [OPTIONS] COMMAND [ARGS]...

  Return information from a PURL.

Options:
  --help  Show this message and exit.

Commands:
  metadata  Given one or more PURLs, for each PURL, return a mapping of...
  urls      Given one or more PURLs, for each PURL, return a list of all...
  validate  Check the syntax of one or more PURLs.
  versions  Given one or more PURLs, return a list of all known versions...
```

And the following subcommands:

- Validate a PURL:

```
$ purlcli validate --help
Usage: purlcli validate [OPTIONS]

```

(continues on next page)

(continued from previous page)

```
  Check the syntax of one or more PURLs.

Options:
  --purl TEXT        PackageURL or PURL.
  --output FILENAME  Write validation output as JSON to FILE.  [required]
  --file FILENAME    Read a list of PURLs from a FILE, one per line.
  --help             Show this message and exit.
```

- Collect package versions for a PURL:

```
$ purlcli versions  --help
Usage: purlcli versions [OPTIONS]

  Given one or more PURLs, return a list of all known versions for each PURL.

  Version information is not needed in submitted PURLs and if included will be
  removed before processing.

Options:
  --purl TEXT        PackageURL or PURL.
  --output FILENAME  Write versions output as JSON to FILE.  [required]
  --file FILENAME    Read a list of PURLs from a FILE, one per line.
  --help             Show this message and exit.
```

- Collect package metadata for a PURL:

```
$ purlcli metadata --help
Usage: purlcli metadata [OPTIONS]

  Given one or more PURLs, for each PURL, return a mapping of metadata fetched
  from the fetchcode package.py info() function.

Options:
  --purl TEXT        PackageURL or PURL.
  --output FILENAME  Write meta output as JSON to FILE.  [required]
  --file FILENAME    Read a list of PURLs from a FILE, one per line.
  --unique           Return data only for unique PURLs.
  --help             Show this message and exit.
```

- Collect package URLs for a PURL:

```
$ purlcli urls --help
Usage: purlcli urls [OPTIONS]

  Given one or more PURLs, for each PURL, return a list of all known URLs
  fetched from the packageurl-python purl2url.py code.

Options:
  --purl TEXT        PackageURL or PURL.
  --output FILENAME  Write urls output as JSON to FILE.  [required]
  --file FILENAME    Read a list of PURLs from a FILE, one per line.
  --unique           Return data only for unique PURLs.
  --head             Validate each URL's existence with a head request.
```

(continues on next page)

(continued from previous page)

```
  --help              Show this message and exit.
```

3.1.3 Funding

This project was funded through the NGI Assure Fund https://nlnet.nl/assure, a fund established by NLnet https://nlnet.nl/ with financial support from the European Commission's Next Generation Internet programme, under the aegis of DG Communications Networks, Content and Technology under grant agreement No 957073.

This project is also funded through grants from the Google Summer of Code program, continuing support and sponsoring from nexB Inc. and generous donations from multiple sponsors.

3.1.4 License

CHAPTER

FOUR

MATCHCODE

Matchcode has the functionalities to index archives, files and directories for purldb packages and API endpoints to make matching available. A ScanCode.io pipeline for matching is also present to match scanned codebases.

4.1 Matchcode

4.1.1 Matchcode functionality

4.1.2 Matching pipeline

CHAPTER

FIVE

HOW-TO DOCUMENTS

How-To documents explain how to accomplish specific tasks.

5.1 How-To-Guides

Here are the various how-to guides across various purldb projects to guide you thourgh specifica use cases:

- Code matching with Matchcode
- Getting symbols from a package (or a PURL)

5.1.1 How to use Matchcode

Instructions on setting up the matchcode server.

5.1.2 How To get symbols and strings from a PURL/package

In this tutorial we'll introduce the different addon pipeline that can be used for collecting symbols and strings from codebase resources.

Note: This tutorial assumes that you have a working installation of PurlDB. If you don't, please refer to the installation page.

Through out this tutorial we will use `pkg:github/llvm/llvm-project@10.0.0` and will show the symbol and string for llvm-project/clang/lib/Basic/Targets/BPF.cpp resource.

```
//===--- BPF.cpp - Implement BPF target feature support -------------------===//
//
// Part of the LLVM Project, under the Apache License v2.0 with LLVM Exceptions.
// See https://llvm.org/LICENSE.txt for license information.
// SPDX-License-Identifier: Apache-2.0 WITH LLVM-exception
//
//===----------------------------------------------------------------------===//
//
// This file implements BPF TargetInfo objects.
//
//===----------------------------------------------------------------------===//
```

(continues on next page)

(continued from previous page)

```
#include "BPF.h"
#include "Targets.h"
#include "clang/Basic/MacroBuilder.h"
#include "clang/Basic/TargetBuiltins.h"
#include "llvm/ADT/StringRef.h"

using namespace clang;
using namespace clang::targets;

const Builtin::Info BPFTargetInfo::BuiltinInfo[] = {
#define BUILTIN(ID, TYPE, ATTRS)                                               \
  {#ID, TYPE, ATTRS, nullptr, ALL_LANGUAGES, nullptr},
#include "clang/Basic/BuiltinsBPF.def"
};

void BPFTargetInfo::getTargetDefines(const LangOptions &Opts,
                                    MacroBuilder &Builder) const {
  Builder.defineMacro("__bpf__");
  Builder.defineMacro("__BPF__");
}

static constexpr llvm::StringLiteral ValidCPUNames[] = {"generic", "v1", "v2",
                                                        "v3", "probe"};

bool BPFTargetInfo::isValidCPUName(StringRef Name) const {
  return llvm::find(ValidCPUNames, Name) != std::end(ValidCPUNames);
}

void BPFTargetInfo::fillValidCPUList(SmallVectorImpl<StringRef> &Values) const {
  Values.append(std::begin(ValidCPUNames), std::end(ValidCPUNames));
}

ArrayRef<Builtin::Info> BPFTargetInfo::getTargetBuiltins() const {
  return llvm::makeArrayRef(BuiltinInfo, clang::BPF::LastTSBuiltin -
                                            Builtin::FirstTSBuiltin);
}
```

Ctags Symbols

- Send GET request to PurlDB with:

```
/api/collect/?purl=pkg:github/llvm/llvm-project@10.0.0&addon_pipelines=collect_
↪symbols_ctags
```

Warning: The `collect_symbols_ctags` pipeline requires `universal-ctags`.

- Once the indexing has completed visit `/api/resources/?purl=pkg:github/llvm/llvm-project@10.0.0` to get the `source_symbols` for resources.

Listing 1: Ctags symbol for `clang/lib/Basic/Targets/BPF.cpp` in `extra_data` field

```json
{
    "package": "http://127.0.0.1:8001/api/packages/<package-id>",
    "purl": "pkg:github/llvm/llvm-project@10.0.0",
    "path": "llvm-project-llvmorg-10.0.0.tar.gz-extract/llvm-project-llvmorg-10.0.0/clang/lib/Basic/Targets/BPF.cpp",
    "type": "file",
    "name": "BPF.cpp",
    "extension": ".cpp",
    "size": 1788,
    "md5": "382b406d1023d12cd8f28106043774ee",
    "sha1": "366146c8228c4e2cd46c47618fa3211ce48d96e2",
    "sha256": "d7609c502c7d462dcee1b631a80eb765ad7d10597991d88c3d4cd2ae0370eeba",
    "sha512": null,
    "git_sha1": null,
    "mime_type": "text/x-c",
    "file_type": "C source, ASCII text",
    "programming_language": "C++",
    "is_binary": false,
    "is_text": true,
    "is_archive": false,
    "is_media": false,
    "is_key_file": false,
    "detected_license_expression": "",
    "detected_license_expression_spdx": "",
    "license_detections": [],
    "license_clues": [],
    "percentage_of_license_text": null,
    "copyrights": [],
    "holders": [],
    "authors": [],
    "package_data": [],
    "emails": [],
    "urls": [],
    "extra_data": {
        "source_symbols": [
            "BUILTIN",
            "BuiltinInfo",
            "ValidCPUNames",
            "fillValidCPUList",
            "getTargetBuiltins",
            "getTargetDefines",
            "isValidCPUName"
        ]
    }
}
```

Xgettext Strings

- Send GET request to PurlDB with:

```
/api/collect/?purl=pkg:github/llvm/llvm-project@10.0.0&addon_pipelines=collect_
↪strings_gettext
```

Warning: The `collect_strings_gettext` pipeline requires `gettext`.

- Once the indexing has completed visit `/api/resources/?purl=pkg:github/llvm/llvm-project@10.0.0` to get the `source_strings` for resources.

Listing 2: Xgettext strings for `clang/lib/Basic/Targets/BPF.cpp` in `extra_data` field

```
{
    "package": "http://127.0.0.1:8001/api/packages/<package-id>",
    "purl": "pkg:github/llvm/llvm-project@10.0.0",
    "path": "llvm-project-llvmorg-10.0.0.tar.gz-extract/llvm-project-llvmorg-10.0.0/
↪clang/lib/Basic/Targets/BPF.cpp",
    "type": "file",
    "name": "BPF.cpp",
    "extension": ".cpp",
    "size": 1788,
    "md5": "382b406d1023d12cd8f28106043774ee",
    "sha1": "366146c8228c4e2cd46c47618fa3211ce48d96e2",
    "sha256": "d7609c502c7d462dcee1b631a80eb765ad7d10597991d88c3d4cd2ae0370eeba",
    "sha512": null,
    "git_sha1": null,
    "mime_type": "text/x-c",
    "file_type": "C source, ASCII text",
    "programming_language": "C++",
    "is_binary": false,
    "is_text": true,
    "is_archive": false,
    "is_media": false,
    "is_key_file": false,
    "detected_license_expression": "",
    "detected_license_expression_spdx": "",
    "license_detections": [],
    "license_clues": [],
    "percentage_of_license_text": null,
    "copyrights": [],
    "holders": [],
    "authors": [],
    "package_data": [],
    "emails": [],
    "urls": [],
    "extra_data": {
        "source_strings": [
            "__bpf__",
            "__BPF__",
```

(continues on next page)

(continued from previous page)

```
            "generic",
            "v",
            "v",
            "v",
            "probe"
        ]
    }
}
```

Tree-Sitter Symbols and Strings

- Send GET request to PurlDB with:

```
/api/collect/?purl=pkg:github/llvm/llvm-project@10.0.0&addon_pipelines=collect_
↪symbols_tree_sitter
```

- Once the indexing has completed visit `/api/resources/?purl=pkg:github/llvm/llvm-project@10.0.0` to get the `source_symbols` and `source_strings` for resources.

Listing 3: Tree-Sitter symbols and strings for `clang/lib/Basic/Targets/BPF.cpp` in `extra_data` field

```
{
    "package": "http://127.0.0.1:8001/api/packages/<package-id>",
    "purl": "pkg:github/llvm/llvm-project@10.0.0",
    "path": "llvm-project-llvmorg-10.0.0.tar.gz-extract/llvm-project-llvmorg-10.0.0/
↪clang/lib/Basic/Targets/BPF.cpp",
    "type": "file",
    "name": "BPF.cpp",
    "extension": ".cpp",
    "size": 1788,
    "md5": "382b406d1023d12cd8f28106043774ee",
    "sha1": "366146c8228c4e2cd46c47618fa3211ce48d96e2",
    "sha256": "d7609c502c7d462dcee1b631a80eb765ad7d10597991d88c3d4cd2ae0370eeba",
    "sha512": null,
    "git_sha1": null,
    "mime_type": "text/x-c",
    "file_type": "C source, ASCII text",
    "programming_language": "C++",
    "is_binary": false,
    "is_text": true,
    "is_archive": false,
    "is_media": false,
    "is_key_file": false,
    "detected_license_expression": "",
    "detected_license_expression_spdx": "",
    "license_detections": [],
    "license_clues": [],
    "percentage_of_license_text": null,
    "copyrights": [],
    "holders": [],
```

(continues on next page)

(continued from previous page)

```
      "authors": [],
      "package_data": [],
      "emails": [],
      "urls": [],
      "extra_data": {
          "source_symbols": [
              "clang",
              "targets",
              "BuiltinInfo",
              "BUILTIN",
              "ID",
              "TYPE",
              "ATTRS",
              "TYPE",
              "ATTRS",
              "ALL_LANGUAGES",
              "getTargetDefines",
              "Opts",
              "Builder",
              "Builder",
              "Builder",
              "ValidCPUNames",
              "isValidCPUName",
              "Name",
              "find",
              "ValidCPUNames",
              "Name",
              "end",
              "ValidCPUNames",
              "fillValidCPUList",
              "Values",
              "Values",
              "begin",
              "ValidCPUNames",
              "end",
              "ValidCPUNames",
              "getTargetBuiltins",
              "makeArrayRef",
              "BuiltinInfo",
              "LastTSBuiltin",
              "FirstTSBuiltin"
          ],
          "source_strings": [
              "BPF.h",
              "Targets.h",
              "clang/Basic/MacroBuilder.h",
              "clang/Basic/TargetBuiltins.h",
              "llvm/ADT/StringRef.h",
              "clang/Basic/BuiltinsBPF.def",
              "__bpf__",
              "__BPF__",
              "generic",
```

(continues on next page)

(continued from previous page)

```
            "v1",
            "v2",
            "v3",
            "probe"
        ]
    }
}
```

Pygments Symbols and Strings

- Send GET request to PurlDB with:

  ```
  /api/collect/?purl=pkg:github/llvm/llvm-project@10.0.0&addon_pipelines=collect_
  ↪symbols_pygments
  ```

- Once the indexing has completed visit `/api/resources/?purl=pkg:github/llvm/llvm-project@10.0.0` to get the `source_symbols` and `source_strings` for resources.

Listing 4: Pygments symbols and strings for `clang/lib/Basic/Targets/BPF.cpp` in `extra_data` field

```
{
    "package": "http://127.0.0.1:8001/api/packages/<package-id>",
    "purl": "pkg:github/llvm/llvm-project@10.0.0",
    "path": "llvm-project-llvmorg-10.0.0.tar.gz-extract/llvm-project-llvmorg-10.0.0/
↪clang/lib/Basic/Targets/BPF.cpp",
    "type": "file",
    "name": "BPF.cpp",
    "extension": ".cpp",
    "size": 1788,
    "md5": "382b406d1023d12cd8f28106043774ee",
    "sha1": "366146c8228c4e2cd46c47618fa3211ce48d96e2",
    "sha256": "d7609c502c7d462dcee1b631a80eb765ad7d10597991d88c3d4cd2ae0370eeba",
    "sha512": null,
    "git_sha1": null,
    "mime_type": "text/x-c",
    "file_type": "C source, ASCII text",
    "programming_language": "C++",
    "is_binary": false,
    "is_text": true,
    "is_archive": false,
    "is_media": false,
    "is_key_file": false,
    "detected_license_expression": "",
    "detected_license_expression_spdx": "",
    "license_detections": [],
    "license_clues": [],
    "percentage_of_license_text": null,
    "copyrights": [],
    "holders": [],
    "authors": [],
```

(continues on next page)

(continued from previous page)

```
      "package_data": [],
      "emails": [],
      "urls": [],
      "extra_data": {
          "source_symbols": [
              "clang",
              "clang",
              "targets",
              "BPFTargetInfo::getTargetDefines",
              "BPFTargetInfo::isValidCPUName",
              "BPFTargetInfo::fillValidCPUList"
          ],
          "source_strings": [
              "\"",
              "__bpf__",
              "\"",
              "\"",
              "__BPF__",
              "\"",
              "\"",
              "generic",
              "\"",
              "\"",
              "v1",
              "\"",
              "\"",
              "v2",
              "\"",
              "\"",
              "v3",
              "\"",
              "\"",
              "probe",
              "\""
          ]
      }
  }
```

CHAPTER SIX

INDICES AND TABLES

- genindex
- modindex
- search