Buildography v1.2.3

Описание ПО и руководство пользователя

2024-11-20

Содержание

Общие сведения

Buildography — инструмент для идентификации реквизитов сборки: файлов программ на входных языках, зависимостей (системных библиотек, заголовочных файлов и т.п.), трансляторов и их конфигурационных файлов.

Предполагаемый сценарий применения — получение структурированной информации о выполнении сценария сборки программного обеспечения в целях инспектирования или аудита. При этом инструмент может быть использован для трассировки произвольных процессов в Linux.

Buildography отслеживает системные вызовы дочернего дерева процессов с помощью системного вызова ptrace и собирает следующую информацию:

- Рабочая директория;
- Аргументы командной строки порождаемых процессов;
- Данные о файлах, которые открывались этими процессами;
- Информация, идентифицирующая вызываемые программы.

Инструмент записывает её в выходной файл в формате JSON.

Этот файл может быть использован для анализа происхождения файлов, возникших в результате сборки: из каких исходных файлов и с помощью каких инструментов они были получены. Некоторые средства анализа входят в состав поставки инструмента.

Руководство пользователя

Запуск инструмента

Утилита командной строки `buildography` имеет интерфейс, аналогичный классическим отладчикам и трассировщикам. Она принимает в качестве аргументов программу и аргументы, с которыми её необходимо выполнить.

Программа может быть задана абсолютным или относительным путём до исполняемого файла, либо только именем. В последнем случае исполняемый файл ищется в директориях, перечисленных в значении переменной окружения `PATH` (значение по умолчанию см. в документации функции `execvp`).

```
buildography [options...] <program> [args...]
```

Утилита принимает следующие опции:

- `-p | --postprocessor <PATH>` — путь до постпроцессора из состава инструмента. Опция актуальна для запуска без установки (при разработке и отладке самого инструмента). Если инструмент уже установлен, опция не требуется.
- `-d | --dump-dir <DIR>` — директория, куда будет помещён выходной файл. По умолчанию используется текущая директория.
- `-f | --filter-subtree <DIR>` — ограничить множество `input` файлами, находящимися в поддереве директории `<DIR>`.
- `-T | --threads <NUM>` — ограничить максимальное число потоков.

Пример

Открытое ПО Postgres можно собрать, выполнив два шага: конфигурацию с помощью скрипта `configure` и сборки с помощью утилиты `make`.

Для перехвата всего процесса сборки и получения результатов в едином JSON файле нужно каким-либо способом объединить все команды сборки. Например, с помощью shell-скрипта.

```
git clone https://github.com/postgres/postgres.git
cd postgres

cat <<EOF > build.sh
#!/bin/bash -e
mkdir -p build
cd build
../configure --without-readline
make -j16
EOF
chmod +x build.sh

/opt/buildography/bin/buildography ./build.sh
```

Другой подход состоит в раздельном анализе шагов сборки. Поскольку в данном примере два шага, запустим инструмент дважды:

```
mkdir -p build2
cd build2
/opt/buildography/bin/buildography ../configure --without-readline
/opt/buildography/bin/buildography make -j16
```

В результате получим два JSON файла.

Формат выходного файла

Результат работы инструмента — единственный `.json` файл с информацией о процессах, порождённых в процессе сборки. Для удобства помимо сырой информации об индивидуальных процессах он содержит уже сгруппированные данные, полученные на её основе — поля `input` и `utilities`. Смысл полей раскрывается ниже.

Выходной JSON файл имеет следующую структуру:

```
{
    "directory": "<PATH>",
    "input": { <FILESPEC...> },
    "commands": [ <ARGV...> ],
    "component_commands": [
        {
            "command": [ <ARGV...> ],
            "dependencies": { <FILESPEC...> },
            "output": { <FILESPEC...> }
        },
        ...
    ],
    "utilities": [
        {
            "path": "<PATH>",
            "hash": "<HASH>"
        },
        ...
    ]
}
```

Где

- `<FILESPEC> ::= "<PATH>": "<HASH>"`;
- `<PATH>` — абсолютный путь к файлу или директории (строка);
- `<HASH>` — хэш-сумма в шестнадцатеричном представлении (в данный момент используется 256-битный вариант ГОСТ Р 34.11-2012 — строка длины 64);
- `<ARGV>` — аргумент командной строки (строка).

Значение полей выходного файла

- `directory` — рабочая директория, из которой была запущена исходная команда.
- `commands` — массив аргументов исходной команды (включая исполняемый файл — `argv[0]`).
- `component_commands` — массив объектов, описывающих вызовы программ, выполненные в процессе выполнения исходной команды. Вложенные вызовы также содержатся в этом массиве, но сам массив — «плоский». Каждый объект содержит следующие поля:
  - `command` — массив аргументов командной строки (аналогично `commands`);
  - `dependencies` — объект, ключи которого — входные файлы для данного запуска программы, а значения — хэш-суммы их содержимого.
  - `output` — объект того же формата, но для выходных файлов.
- `input` — объект, ключи которого — входные файлы (полученные обходом `dependencies` всех `component_commands` с учётом фильтрации `--filter-subtree`), а значения — хэш-суммы их содержимого.
- `utilities` — массив объектов, описывающих все вызванные программы. Каждый объект имеет два поля:
  - `path` — путь до исполняемого файла программы;
  - `hash` — хэш-сумма содержимого этого файла.

Контрольные суммы

Для идентификации файлов используются контрольные суммы (хэш-суммы) их содержимого. Они вычисляются с помощью хэш-функции, определённой ГОСТ 34.11-2018 (ГОСТ Р 34.11-2012).

Название выходного файла составлено из хэш-суммы его содержимого и `.json`.

Средства анализа

Выходной JSON файл можно анализировать как вручную, так и создавая инструменты для автоматического исследования свойств сборки. Несколько таких инструментов входят в состав поставки buildography.

Сторонние бинарные файлы — `bxy-find-unbuilt-binaries`

Скрипт `bxy-find-unbuilt-binaries` выводит в файл перечень использованных при сборке «бинарных» файлов, *которые не были созданы в процессе этой сборки.* Мотивация применения — убедиться в том, что ПО собирается целиком из исходных файлов или выявить предсобранные зависимости.

```
bxy-find-unbuilt-binaries -i INPUT_FILE -o OUTPUT_FILE
                          [--is-binary-pred PREDICATE]
```

Под «бинарными» файлами по умолчанию понимаются файлы формата ELF и ar архивы. Эта проверка реализуется функцией `is_binary`, определённой в самом скрипте. Пользователь имеет возможность определить собственный предикат (команду, принимающую единственный аргумент — путь к файлу) для ответа на вопрос, является ли файл бинарным.

Пример запуска:

```
json=8b5177e2bd383896983965e793bee81c19a358068d0d84460442603edc715979.json
  bxy-find-unbuilt-binaries -i "$json" -o blobs.txt
```

В результате для примера сборки postgres `blobs.txt` содержит (приводится сокращённый список):

```
/usr/bin/install
/usr/lib/bfd-plugins/libdep.so
/usr/lib/crti.o
/usr/lib/gcc/x86_64-pc-linux-gnu/14.2.1/libgcc.a
/usr/lib/gcc/x86_64-pc-linux-gnu/14.2.1/liblto_plugin.so
/usr/lib/ld-linux-x86-64.so.2
/usr/lib/libacl.so.1.1.2302
/usr/lib/libc_nonshared.a
/usr/lib/libcrypt.so.2.0.0
/usr/lib/libc.so.6
/usr/lib/libguile-3.0.so.1.7.0
/usr/lib/libicudata.so.75.1
/usr/lib/libLLVM.so.18.1
/usr/lib/liblzma.so.5.6.2
/usr/lib/libmpfr.so.6.2.1
/usr/lib/libm.so.6
/usr/lib/libstdc++.so.6.0.33
/usr/lib/libxml2.so.2.13.3
/usr/lib/libz.so.1.3.1
/usr/lib/libzstd.so.1.5.6
/usr/lib/LLVMgold.so
/usr/lib/perl5/5.38/core_perl/auto/Cwd/Cwd.so
/usr/lib/perl5/5.38/core_perl/CORE/libperl.so
/usr/lib/Scrt1.o
...
```

Внешние исходные файлы — `bxy-find-external-sources`

Скрипт `bxy-find-external-sources` выводит в файл перечень файлов, использованных в процессе сборки и при этом:

1. не находящихся в заданных каталогах `SRC_DIR` и их подкаталогах;
2. не созданных в результате выполнения команд сборки;
3. являющихся «исходниками».

```
bxy-find-external-sources -i INPUT_FILE -o OUTPUT_FILE -d SRC_DIR [ -d ... ]
                          [--is-source-pred PREDICATE]
```

Под «исходниками» по умолчанию понимаются файлы, для которых выполнен предикат `is_source`. По умолчанию он определён как отрицание предиката `is_binary` (см. подраздел выше). Пользователь имеет возможность переопределить предикат `is_source` произвольной командой, принимающей единственный аргумент — путь к файлу.

Мотивация применения — убедиться, что все исходные коды в процессе сборки извлекаются из заданных каталогов или выявить внешние по отношению к ним файлы с исходным кодом.

Пример запуска:

```
bxy-find-external-sources \
    -i 8b5177e2bd383896983965e793bee81c19a358068d0d84460442603edc715979.json \
    -o external-src.txt \
    -d /mnt/co/postgres \
    -d /usr/lib \
    -d /usr/include
```

Возможное содержимое `external-src.txt` после запуска скрипта:

```
/etc/nsswitch.conf
/etc/passwd
/proc/602303/maps
...
/proc/628707/maps
/proc/sys/kernel/ngroups_max
/usr/bin/egrep
...
/usr/share/bison/m4sugar/foreach.m4
...
/usr/share/bison/skeletons/c-skel.m4
/usr/share/bison/skeletons/yacc.c
/usr/share/locale/locale.alias
/usr/share/perl5/core_perl/Carp.pm
...
/usr/share/perl5/core_perl/XSLoader.pm
```

Ингредиенты файла — `bxy_trace_origins`

Скрипт `bxy_trace_origins` по выходному файлу строит множество *исходных* файлов, «составляющих» его. В качестве выходного файла могут быть переданы в том числе любые промежуточные файлы, полученные в ходе сборки.

```
usage: bxy_trace_origins [-h] -d JSON_DIRECTORY -a BUILD_ARTIFACT
                         [-o OUTPUT_FILE] --hasher HASHER
                         [-f FILTER_SUBTREE] [--with-intermediate]
                         [--format {plain,pdf}]
```

Обязательные аргументы:

- С помощью `--json-directory` задаётся директория, содержащая JSON файлы.
- С помощью `--build-artifact` задаётся путь до артефакта сборки в файловой системе.
- С помощью `--hasher` задаётся командная строка для запуска утилиты, печатающей хэш-сумму переданного ей файла-агрумента. Этот параметр нужен, чтобы вычислить контрольную сумму переданного файла. Далее для поиска в JSON записей об этом файле он идентифицируется контрольной суммой.

Опциональные аргументы:

- С помощью `--output-file` задаётся путь до файла, в который будет записана информация о зависимостях сборки артефакта.
- С помощью `--filter-subtree` задаётся директория, по которой будут отфильтрованы найденные файлы: файл будет выведен в том случае, если он находится в заданном каталоге или его подкаталоге.
- Если передана опция `--with-intermediate`, скрипт помимо найденных исходных файлов выводит ещё и промежуточные зависимости.
- С помощью `--format` задаётся формат отчёта о зависимостях сборки артефакта. Доступны следующие форматы: обычный текст, PDF.

Пример запуска:

```
bxy_trace_origins \
    --json-directory . \
    --hasher "gost12-256-hash" \
    --build-artifact build/src/pl/plpgsql/src/plpgsql.so \
    --output-file trace-origins.txt
```

Возможное содержимое `trace-origins.txt` после запуска скрипта:

```
748d255fc...  /usr/share/locale/locale.alias
550a10d61...  /mnt/co/postgres/src/include/utils/plancache.h
5bdd142c6...  /mnt/co/postgres/src/include/access/xact.h
238ea9ca4...  /usr/lib/libz.so.1.3.1
c7f66e87b...  /usr/include/bits/types/__locale_t.h
1fd07fd0f...  /mnt/co/postgres/src/include/lib/simplehash.h
2753c6ed1...  /usr/include/bits/select.h
4a8dda1f5...  /mnt/co/postgres/src/include/common/kwlookup.h
c707bf1c1...  /mnt/co/postgres/src/include/access/tupmacs.h
b1bb13531...  /usr/include/bits/stdint-least.h
cdda67733...  /mnt/co/postgres/src/include/storage/procnumber.h
8fa04cb22...  /usr/include/bits/wordsize.h
2793fa39d...  /mnt/co/postgres/src/include/pg_config_ext.h
73e23d526...  /mnt/co/postgres/src/include/storage/relfilelocator.h
bfab053fa...  /mnt/co/postgres/src/include/access/tupdesc.h
5c641f350...  /usr/include/unistd.h
...
```

Поддерживаемые архитектуры и сборочные системы

В настоящее время инструмент работает для AMD64 и ARM64.

Работоспособность протестирована для систем сборки GNU autotools+make, CMake, Meson. Для апробации использовалось программное обеспечение, написанное преимущественно на языках Си и C++.

Руководство администратора

Системные требования

Для использования buildography требуются:

- ядро Linux версии не ниже 4.8, сконфигурированное с `CONFIG_SECCOMP_FILTER=y`
- librhash и утилита командной строки `rhash` версии не ниже 1.3.8
- python3

Средства анализа, входящие в состав поставки, также используют

- bash
- jq
- typst (опционально, для генерации файла с зависимостями сборки в формате PDF)

Установка предсобранного инструмента

Инструмент распространяется в виде готовых бинарных сборок. Для установки из архива следует использовать команду

```
tar xf buildography-amd64.tar.gz -C <install-dir>
```

Принцип работы инструмента

Трассировщик

Трассировщик полагается на использование ptrace API — операций, реализованных в ядре Linux и доступных через системный вызов `ptrace` (конкретная операция определяется параметрами вызова).

Существенная деталь реализации трассировщика — использование `seccomp` для фильтрации системных вызовов, при выполнении которых трассировщик получает управление. BPF программа, определяющая, как обрабатывать системные вызовы, выполняется в контексте ядра, что позволяет существенно снизить накладные расходы.

Для получения необходимой информации трассировщик отслеживает следующие системные вызовы:

Системный вызов	Необходимая информация
`open/openat/openat2`	Путь к файлу и режим, в котором он был открыт
`execve`	Список аргументов команды
`fork/vfork/clone`	Идентификатор нового процесса

Постпроцессор

Трассировщик генерирует промежуточный JSON файл. Оставшаяся часть работы выполняется в постпроцессоре (`bxy_postprocessor`). В неё входит заполнение поля `input` и расширение объектов в массиве `utilities`.

Первым шагом обработки JSON-файла, полученного из трассировщика, является удаление выходных файлов с неподсчитанной хэш-суммой: на данном этапе удаляются записи о временных файлах, которые не были прочитаны в процессе сборки.

Далее происходит генерация множества входных файлов: по умолчанию множеством входных файлов считаются все файлы, которые были открыты на чтение и не были открыты на запись в процессе трассировки.

Последним шагом постпроцессор может дополнить информацию об использованных инструментах. По умолчанию в результирующий файл вносится информация о пути к исполняемому файлу инструмента и хэш-сумма его содержимого. Эта информация копируется из промежуточного JSON,

куда она уже записана трассировщиком. Однако кроме этого в коде постпроцессора предусмотрены точки расширения — функции `get_version`, `get_verbose` и `get_config_files_info`, позволяющие сгенерировать дополнительные поля с информацией об инструментах.