ИСП РАН

# Программное обеспечение «ISPuaro»

## Краткое руководство пользователя

2025 г.

## Содержание

# 1. Описание функциональных возможностей

Программное обеспечение «ISPuaro» (далее – Инструмент «ISPuaro», Инструмент анализа компонентов ПО, альтернативное название – «ISPuaro») применяется для анализа компонентов ПО и сопровождающих артефактов с использованием поиска клонов кода, сопоставления бинарных файлов с исходным кодом, а также с собранной базой знаний, дефектами и метаинформацией. Настоящий документ является кратким руководством пользователя Инструмента «ISPuaro».

Инструмент «ISPuaro» позволяет пользователям проводить инвентаризацию ПО и поиск возможных дефектов. База знаний содержит более 300 тысяч пакетов Debian, 5 миллионов бинарных пакетов и 280 тысяч записей о дефектах и соответствующих им версиях ПО для точного определения дефектов в анализируемой программе. Для увеличения гибкости поиска клонов кода в Инструменте «ISPuaro» реализован функционал добавления пользовательских проектов в базу знаний.

Основными функциями Инструмента «ISPuaro» являются:

- анализ компонентов и инвентаризация ПО;
- поиск возможных дефектов;

Минимальные системные требования: для функционирования Инструмента «ISPuaro» требуется ОС Ubuntu 20.04, процессор архитектуры x86-64 (минимум 4 ядра), 128 ГБ оперативной памяти, размер свободного дискового пространства не менее 3 ТБ.

Инструмент «ISPuaro» разрабатывается на языке программирования Python (версия >=3.9).

# 2. Лицензирование и установка

По вопросам определения стоимости, приобретения и использования обращайтесь по адресу: ispuaro@ispras.ru.

## 2.1 Системные требования

Минимальные системные требования: Для функционирования Инструмента требуется персональный компьютер с архитектурой x86-64, не менее 128 ГБ оперативной памяти, размер свободного дискового пространства не менее 3 ТБ. Поддерживаются следующие 64-разрядные дистрибутивы операционной системы Linux: Ubuntu 20.

Список используемого ПО с указанием лицензий.

| № | Название ПО | Лицензия | Открытый репозиторий |
|---|---|---|---|
| 1 | Neo4j | GNU General Public License version 3 | https://github.com/neo4j/neo4j |
| 2 | NeoDash | Apache License Version 2.0 | https://github.com/neo4j-labs/neodash |
| 3 | Docker | Apache License Version 2.0 | https://github.com/moby/moby |
| 4 | Sbuild | GNU General Public License version 2 | https://salsa.debian.org/debian/sbuild |
| 5 | Checksec | The 3-Clause BSD License | https://salsa.debian.org/debian/checksec |
| 6 | Binmap | Apache License Version 2.0 | https://github.com/quarkslab/binmap (локальный собственный с исправлениями) |

## 2.2 Установка

Для установки Инструмента «ISPuaro» в ОС Ubuntu 20.04 с python версии >= 3.9 необходимо перейти в директорию с «ispuaro». Для удобства рекомендуется использовать виртуальное окружение Python.

```
# Начальная директория
cd /home/user/ispuaro
# Создание и активация окружения
python3.9 -m venv ispuaro_env
source ispuaro_env/bin/activate
# Установка необходимых зависимостей
pip install -r cli/requirements.txt
cd release/app/build
./install_deps.sh
cd ../../../
```

Для работы с инструментом «ISPuaro» требуется:

1. Установить Docker и Docker Compose;
2. Добавить текущего пользователя в группу docker (`sudo usermod -aG docker $USER`).

# 3. Сценарий работы с Инструментом «ISPuaro»

Для ознакомления и работы с Инструментом «ISPuaro» в операционной системе Ubuntu 20.04 необходимо выполнить следующие действия:

1. Запустить текстовый интерфейс «ISPuaro»
2. Выполнить развертывание сервера Инструмента «ISPuaro»
3. Загрузить снимок тестовой базы знаний
4. Загрузить пользовательские проекты в базу знаний
5. Запустить обработку пользовательских проектов
6. Запустить анализ компонентов целевого ПО
7. Просмотреть результаты анализа компонентов

## 3.1 Запуск текстового интерфейса «ISPuaro»

Для работы инструмента «ISPuaro» необходим конфигурационный файл .env. Для данного примера конфигурационный файл .env с настройками параметров развертывания подготовлен заранее, он находится в тестовой директории cli_user_guide. Его необходимо переместить в корневую директорию.

```
cp cli_user_guide/.env ./
```

Далее необходимо запустить текстовый интерфейс (TUI).

```
./cli/cli_main
```

После запуска отобразится текстовое меню управления (см. Рисунок 3.1).


Рисунок 3.1 – Текстовое меню управления

В меню доступны следующие опции:

1. **"Deployment configuration"** – Развертывание (настройка) сервера «ISPuaro»;
2. **"API"** - Выбор функционала «ISPuaro»;
3. **"Exit"** – Выход.

Для выбора нужной опции необходимо ввести соответствующий символ (1, 2 или q). В меню **"Deployment configuration"** представлены возможности развертывания и управления сервером «ISPuaro». Меню "API" содержит список доступных функций анализа.

### 3.2 Развертывание сервера Инструмента «ISPuaro»

Для развертывания сервера необходимо перейти в меню **"Deployment configuration"**, выбрав опцию 1. После перехода отобразится текстовое меню развертывания сервера (см. Рисунок 3.2).

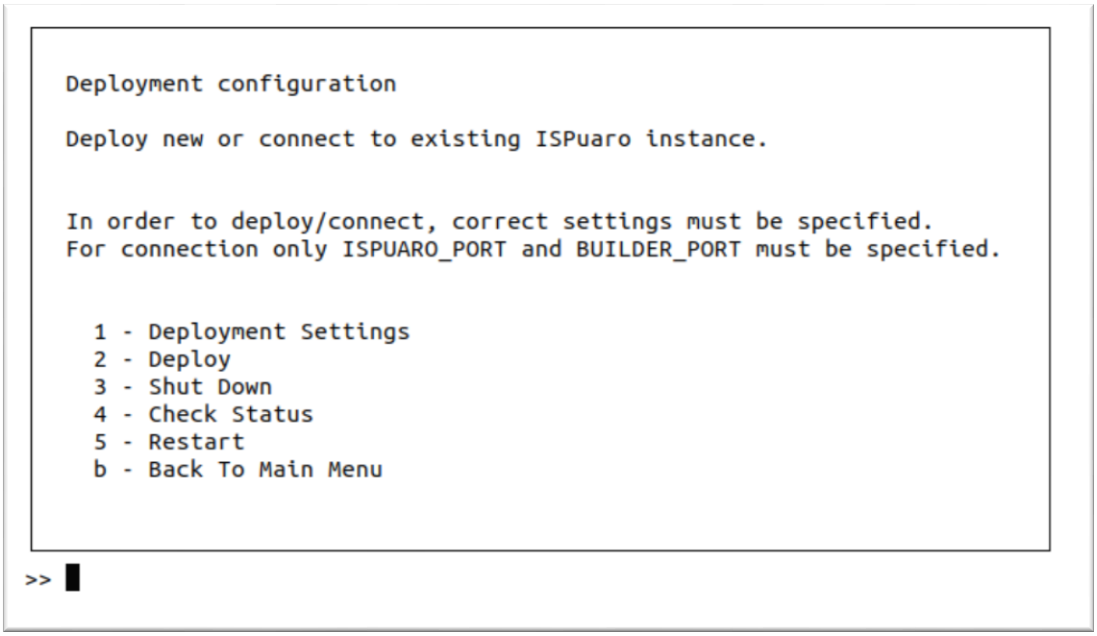


Рисунок 3.2 – Текстовое меню развертывания сервера

В меню доступны следующие опции:

1. **"Deployment settings"** - Настройка параметров развертывания;
2. **"Deploy"** - Развертывание сервера;
3. **"Shut Down"** - Выключение сервера;
4. **"Check Status"** - Проверка доступности сервера;
5. **"Restart"** - Перезапуск сервера;
6. **"Back To Main Menu"** - Возврат в меню управления.

Для настройки параметров развертывания необходимо выбрать опцию 1 - **"Deployment settings"**. В появившемся окне будут представлены параметры для развертывания сервера (см. Рисунок 3.3). Значения параметров по умолчанию берутся из конфигурационного файла .env.

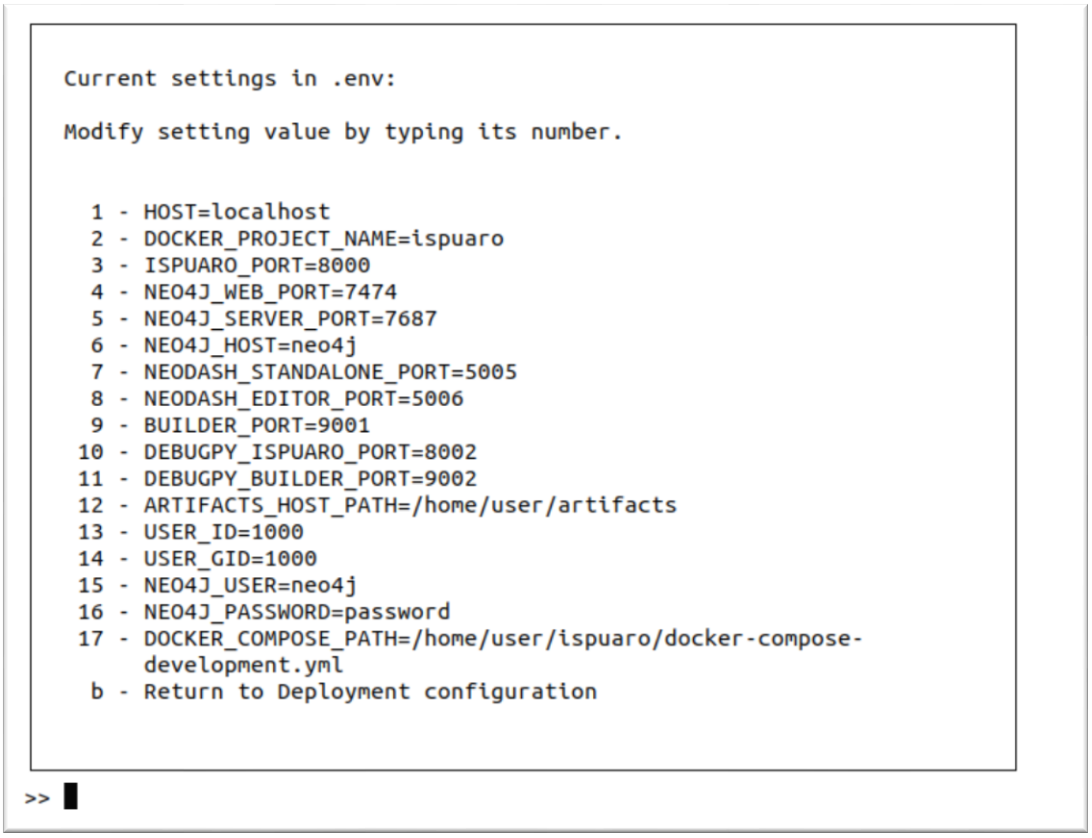


Рисунок 3.3 – Параметры развертывания сервера (представлены значения по умолчанию)

Представлены следующие параметры развертывания:

1. **"HOST"** – Ip-адрес машины, на которой развернут сервер «ISPuaro» (с возможностью подключения к удаленному серверу);
2. **"DOCKER_PROJECT_NAME"** - Имя Docker-группы, под которым запустится ISPuaro (для локального развертывания);
3. **"ISPUARO_PORT"** - Порт, на котором будет работать основной сервер «ISPuaro» (web);
4. **"NEO4J_WEB_PORT"** - Порт, на котором будет работать визуальный интерфейс сервера Neo4j (для локального развертывания);
5. **"NEO4J_SERVER_PORT"** - Порт, на котором будет работать сам сервер Neo4j;
6. **"NEO4J_HOST"** – Ip-адрес машины, на которой работает сервер Neo4j (для локального подключения или удаленного подключения к другой базе);
7. **"NEODASH_STANDALONE_PORT"** - Порт, на котором будет работать информационная панель Neo4j для просмотра визуализации хранимых данных;

8. **"NEODASH_EDITOR_PORT"** - Порт, на котором будет работать информационная панель Neo4j для редактирования отображаемых визуализаций;
9. **"BUILDER_PORT"** - Порт, на котором будет работать сервер сборки и анализа пакетов Debian;
10. **"DEBUGPY_ISPUARO_PORT"** - Порт, с помощью которого можно удаленно отлаживать основной сервер «ISPuaro» (оставить значение по умолчанию);
11. **"DEBUGPY_BUILDER_PORT"** - Порт, с помощью которого можно удаленно отлаживать сервер сборки и анализа пакетов Debian (оставить значение по умолчанию);
12. **"ARTIFACTS_HOST_PATH"** - Путь в файловой системе, где хранятся загруженные пакеты Debian, проекты для анализа и результаты анализа;
13. **"USER_ID"** – Идентификатор пользователя (UID) для развёртывания (локально). Получить: **"**id -u**"**;
14. **"USER_GID"** - Идентификатор группы (GID) пользователя, от имени которого запускаются контейнеры (для локального развёртывания). Получить: **"**id -g**";**
15. **"NEO4J_USER=neo4j"** - Логин для аутентификации на сервере Neo4j (В тестовом примере используется **"**neo4j**"**);
16. **"NEO4J_PASSWORD"** - Пароль для аутентификации на сервере Neo4j (В тестовом примере используется **"**password**"**);
17. **"DOCKER_COMPOSE_PATH"** – Путь до файла конфигурации, например, docker-compose-development.yml, который содержит необходимые инструкции для запуска и настройки сервисов;
18. **"Return to Deployment configuration"** - Возврат в меню развертывания сервера.

Для изменения параметра необходимо выбрать соответствующий номер и ввести желаемое значение.

После настройки параметров, необходимо перейти обратно в меню развертывания сервера и выбрать опцию 2 (**"Deploy"**), чтобы развернуть сервер. После выбора опции появится окно, где будут показаны запуски контейнеров (см. Рисунок 3.4).

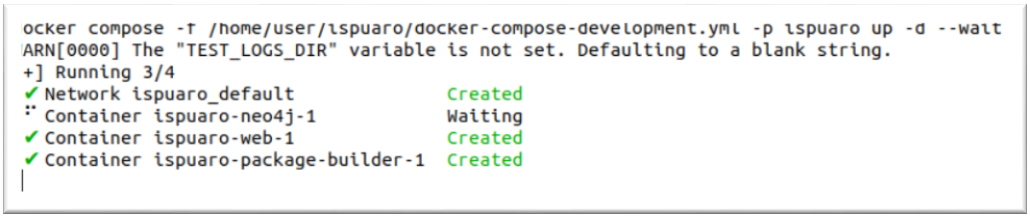


Рисунок 3.4 – Развертывание сервера

Далее из окна можно выйти, развертывание сервера может занять некоторое время. Статус развертывания можно проверить, выбрав опцию 4 **- "Check Status"** (см. Рисунок 3.2). Необходимо дождаться, когда "Web status" и "Builder status" будут иметь статус "Connected", как показано на Рисунке 3.5 (окно необходимо обновлять вручную).

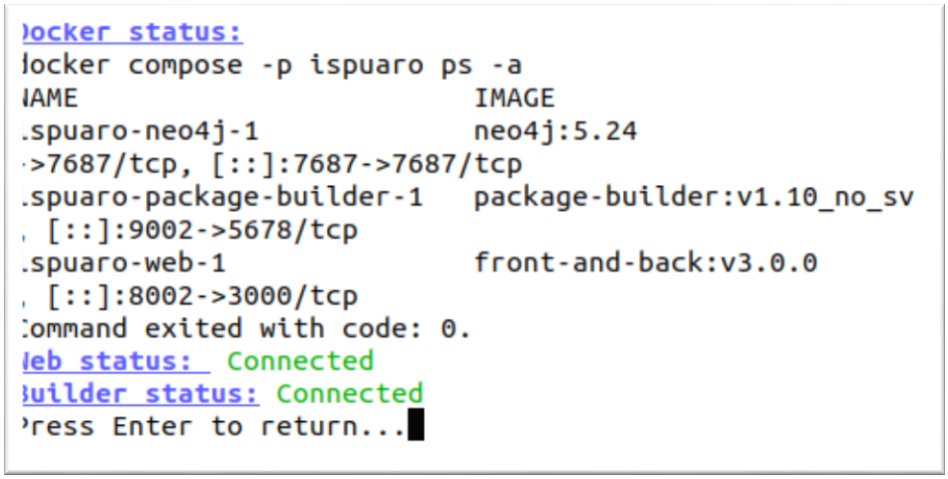


Рисунок 3.5 – Статус развертывания

Информация представлена о следующих статусах:

1. **"Docker status"** – Информация о развернутых локально Docker-сервисах;
2. **"Web status"** - Статус подключения к основному серверу «ISPuaro»;
3. **"Builder status"** - Статус подключения к серверу сборки и анализа пакетов Debian.

Статус на Рисунке 3.5 указывает, что сервер «ISPuaro» и соответствующие контейнеры развернуты и работают корректно. Далее можно приступать к работе с Инструментом «ISPuaro».

Для выключения сервера и контейнеров «ISPuaro» необходимо выбрать опцию 3 - **"Shut Down"** (см. Рисунок 3.2).

Для перезапуска определенного контейнера (или всех) необходимо выбрать опцию 5 - **"Restart"** (см. Рисунок 3.2).

### 3.3 Загрузка снимка тестовой базы знаний

Для работы с Инструментом «ISPuaro» необходимо вернуться в меню управления (см. Рисунок 3.1) и перейти в меню **"API"**, выбрав опцию 2. После перехода отобразится текстовое меню **"API"**, где показан функционал Инструмента «ISPuaro» (см. Рисунок 3.6).

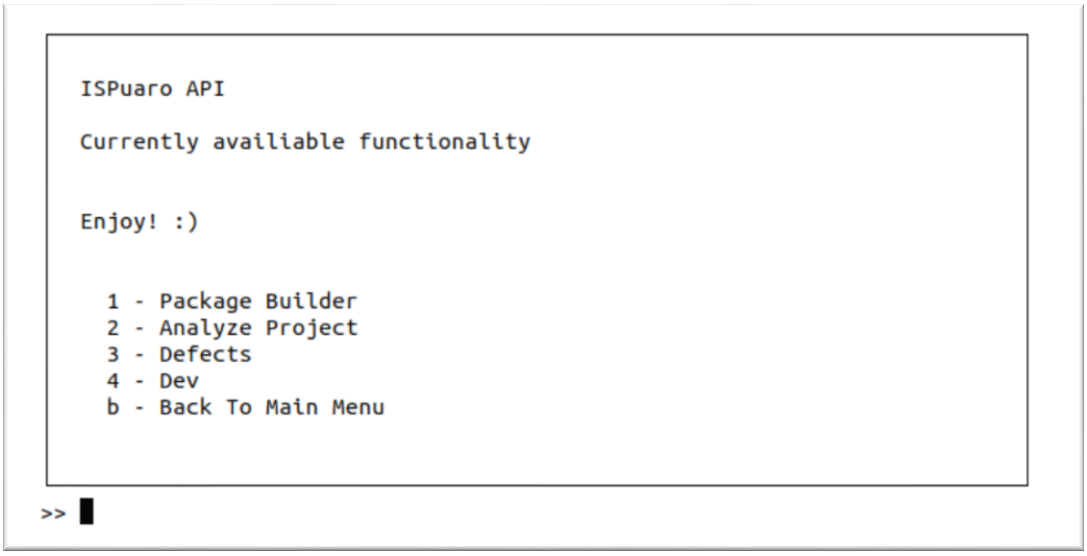


Рисунок 3.6 – Текстовое меню API

Далее необходимо указать путь до снимка загружаемой базы знаний. Для этого нужно выбрать опцию 4 - **"Dev"**. После перехода отобразится меню (см. Рисунок 3.7), в котором необходимо выбрать опцию 1 - **"Import Neo4J Dump"** и указать путь до снимка загружаемой базы знаний, и параметры, которые были указали в пункте 3.2 во время настройки сервера, часть из которых по умолчанию (см. Рисунок 3.8).

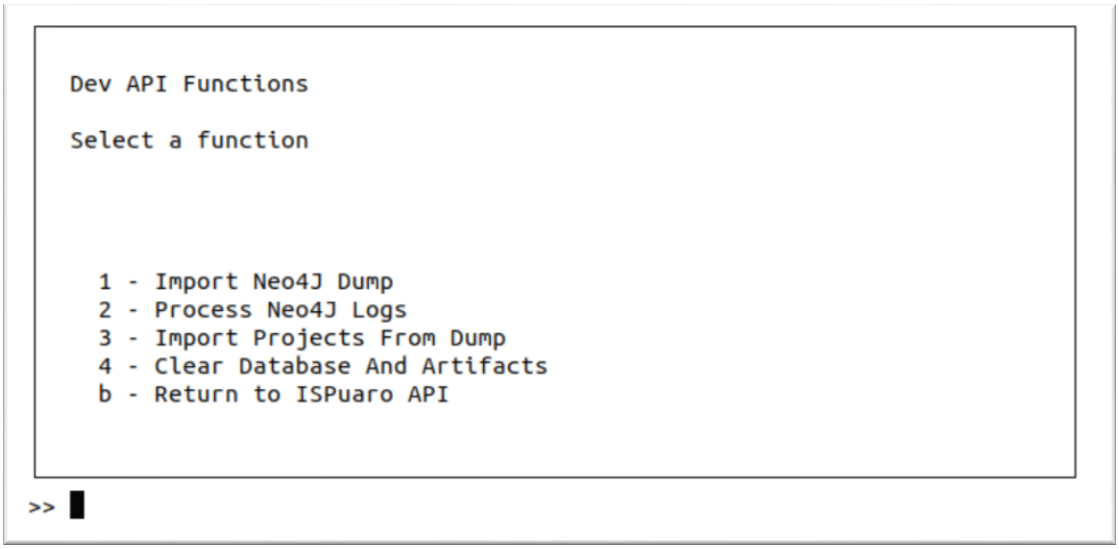


Рисунок 3.7 – Текстовое меню Dev


Рисунок 3.8 – Настройка параметров

Представлены следующие параметры:

1. **"dump" -** путь до загружаемой базы знаний;
2. **"host" -** локальная машина: localhost;

3. **"port"** - порт на локальной машине: 7687;
4. **"name"** - имя пользователя: neo4j:
5. **"password"** - пароль: password;

Для загрузки снимка тестовой базы знаний необходимо выбрать опцию r - **"RUN"**.

Для переноса снимка базы знаний в файловую систему, взаимодействующую с развернутыми Docker-сервисами, необходимо вернуться в меню **"Dev"** и выбрать опцию **"Import Projects From Dump"** (см. Рисунок 3.7). После перехода отобразится меню **"Import Projects From Dump"**, в котором необходимо указать параметры (см. Рисунок 3.9).

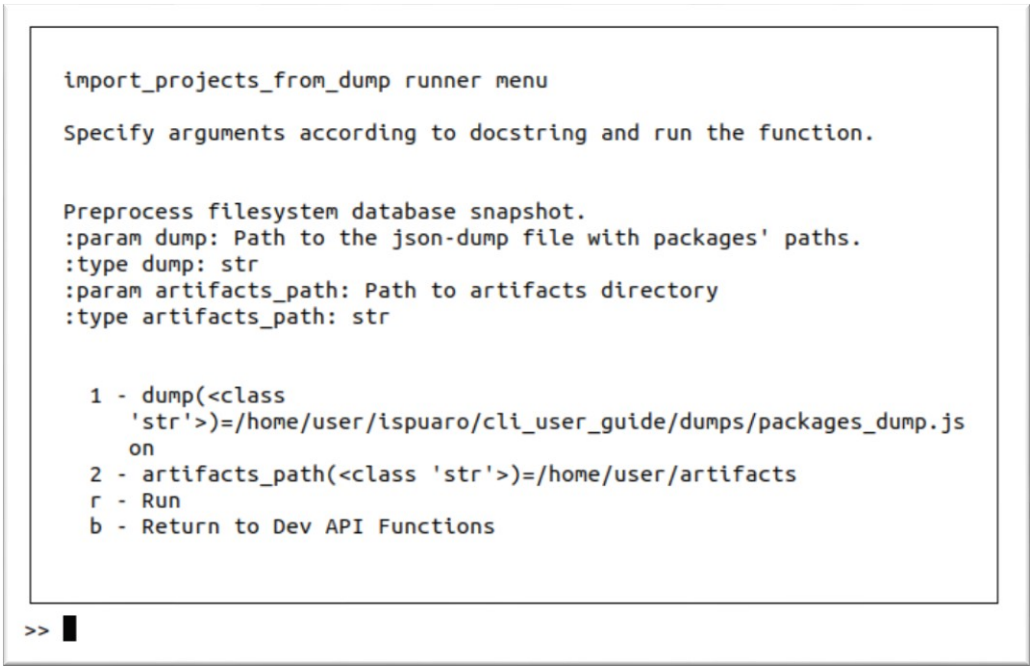


Рисунок 3.9 – Меню Import Projects From Dump

### 3.4 Загрузка пользовательских проектов в базу знаний

Для загрузки пользовательских проектов необходимо вернуться в меню **"API"** и выбрать опцию 1 - **"Package Builder"** (см. Рисунок 3.6). После перехода отобразится меню **"Package Builder"** (см. Рисунок 3.10).

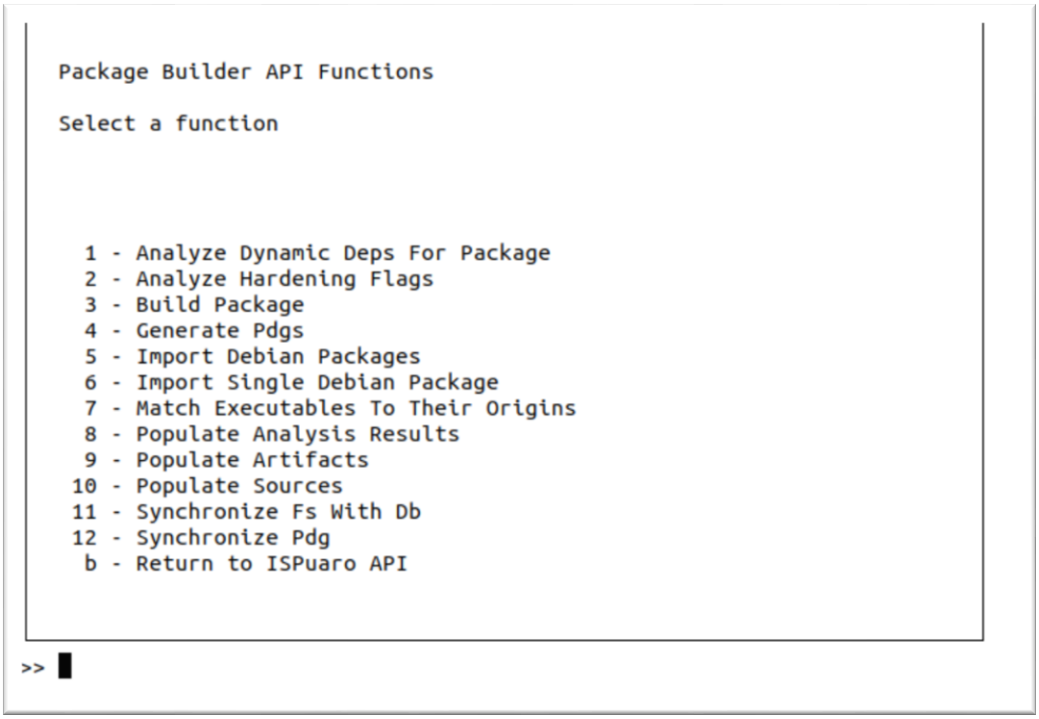


Рисунок 3.10 – Меню Package Builder

Для загрузки пользовательских проектов необходимо выбрать опцию 5 - **"Import Debian Packages"**. Пользовательские проекты должны быть упакованы в tar-архивы. Допускается размещение нескольких проектов в одном tar-архиве. Полученные tar-архивы необходимо упаковать в один общий tar-архив. Если в результате получилось несколько общих tar-архивов, то их необходимо загружать по отдельности.

В меню **"Import Debian Packages"** (см. Рисунок 3.11) необходимо указать путь до общего tar-архива с проектами, выбрав параметр 2 - **"packages_archive_path"** и остальные параметры. После указания пути к архиву действие необходимо подтвердить, нажав Y в терминале.

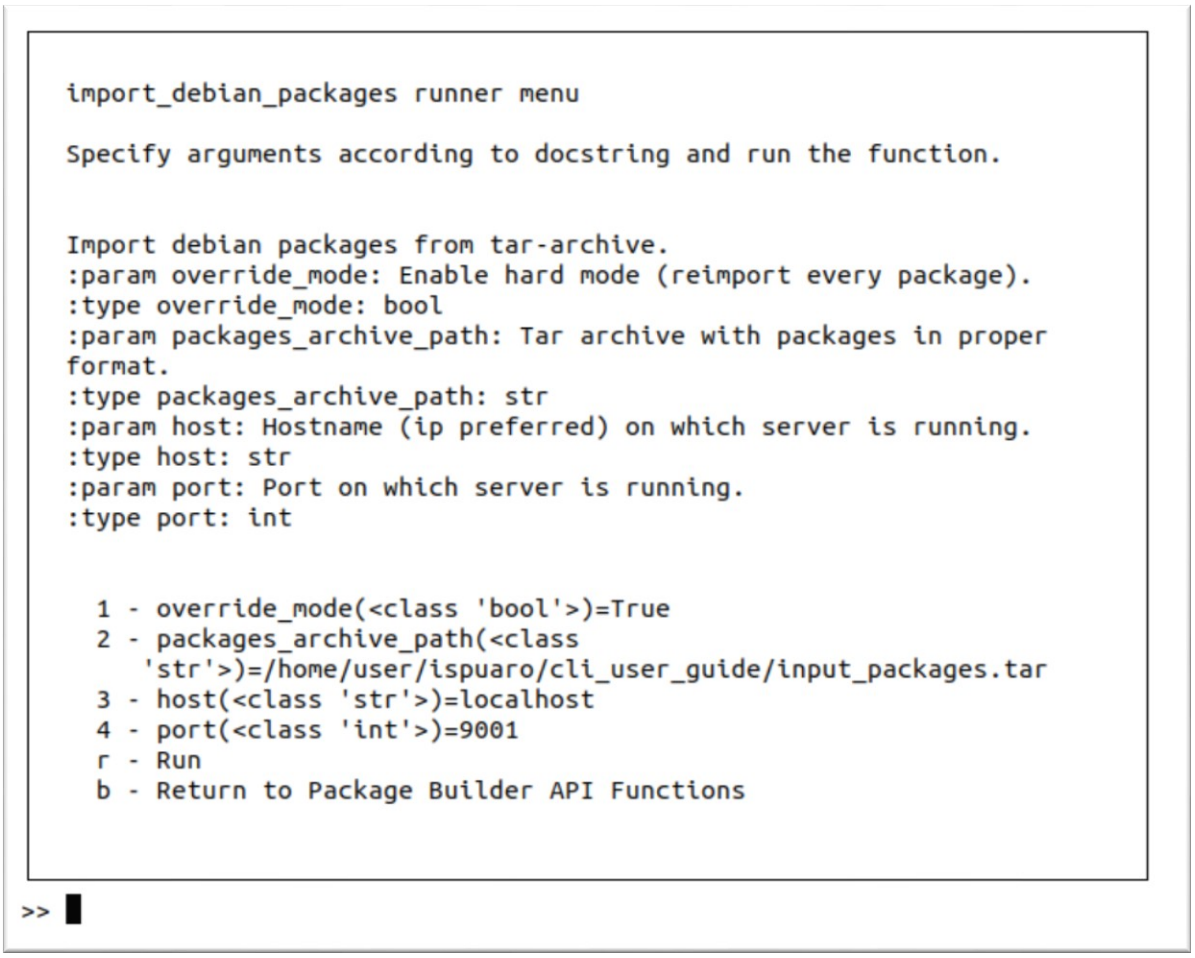


Рисунок 3.11 – Меню Import Debian Packages

Пример визуализации заполнения тестовой базы знаний проектами с открытым исходным кодом (слева вверху - openssl), бинарным кодом (bash, справа вверху) и пользовательскими проектами (внизу - sudo) представлен на Рисунке 3.12.

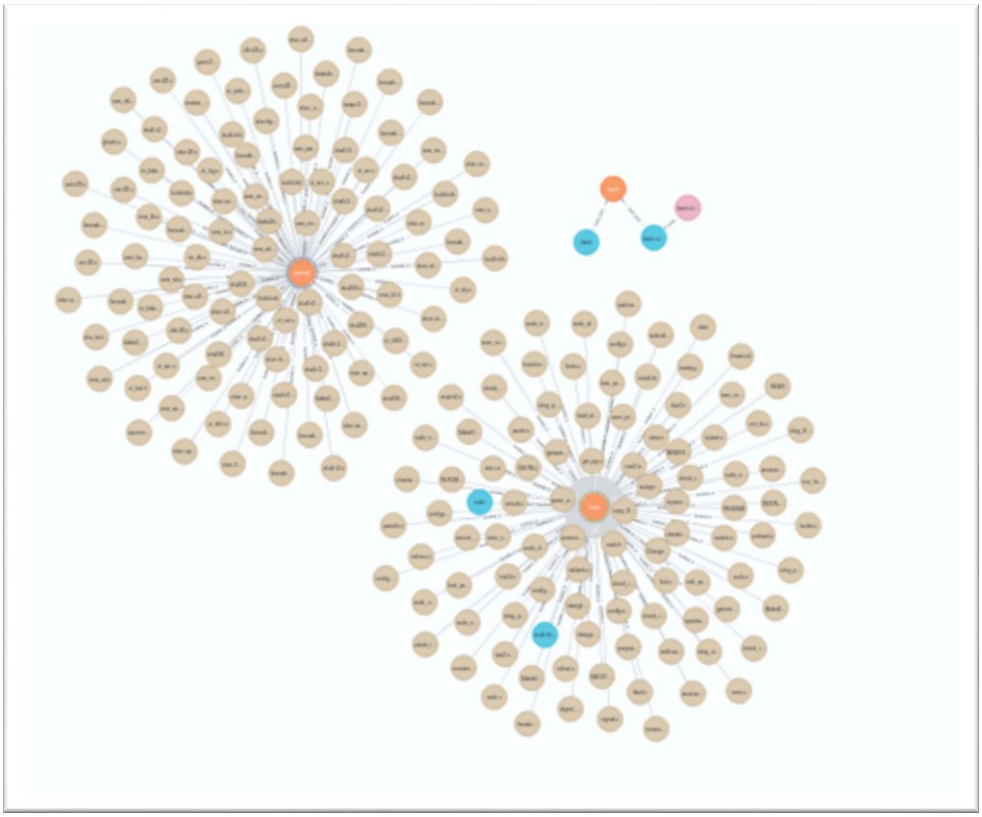

Рисунок 3.12 – Пример визуализации

### 3.5 Запуск обработки пользовательских проектов

После загрузки пользовательского проекта необходимо провести дополнительные действия для его обработки (действия также проводятся в меню **"Package Builder"** (см. Рисунок 3.10), значения опций используются по умолчанию). После каждого действия необходимо выбрать опцию r - **"Run"** и подтвердить операцию, нажав Y в терминале.

1. Выгрузить исходный код в базу знаний с помощью опции 10 - **"Populate Sources"** (см. Рисунок 3.13);

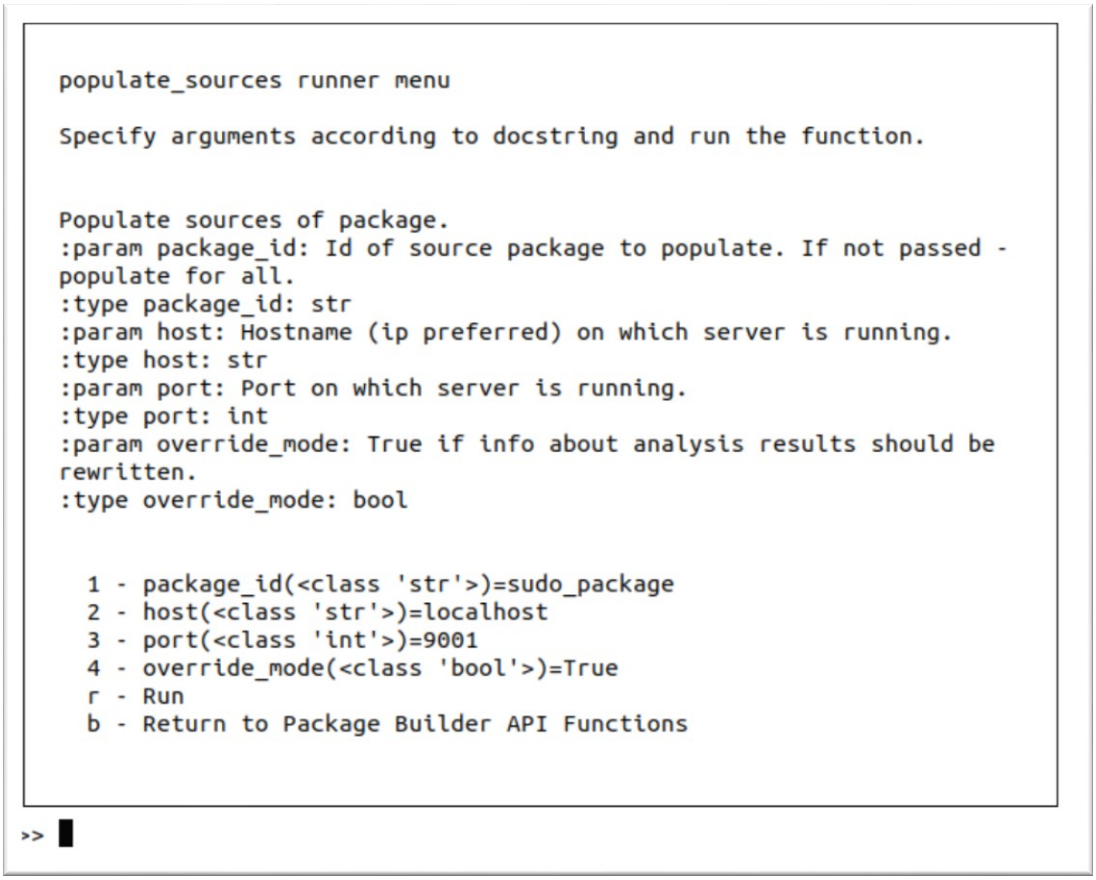


Рисунок 3.13 – Populate Sources

2. Выгрузить артефакты проекта, такие как бинарный код, с помощью опции 9 - **"Populate Artifacts"** (см. Рисунок 3.14);

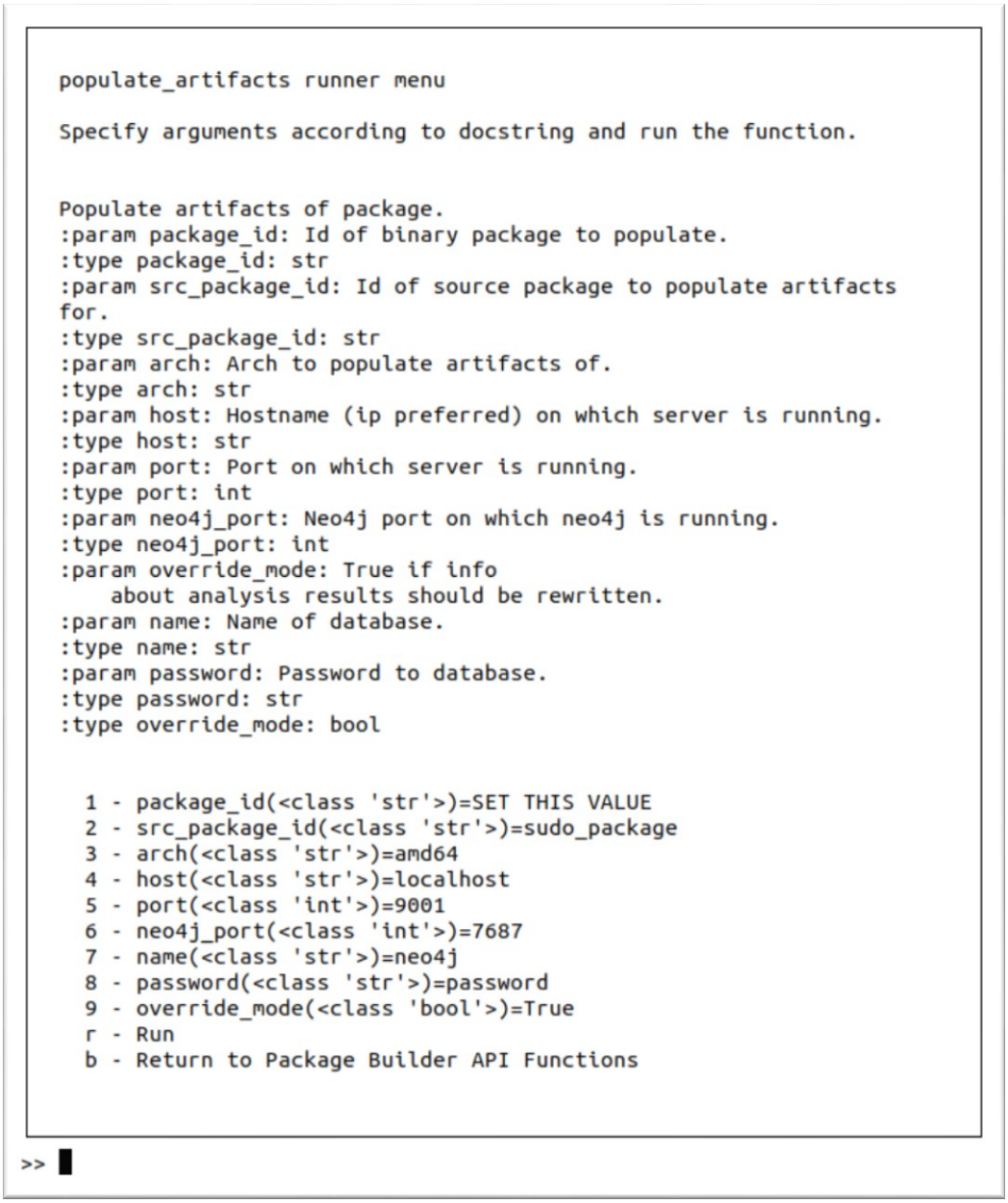


Рисунок 3.14 –Populate Artifacts

3. Сгенерировать PDG с помощью опции 4 - **"Generate PDGs"** (см. Рисунок 3.15);

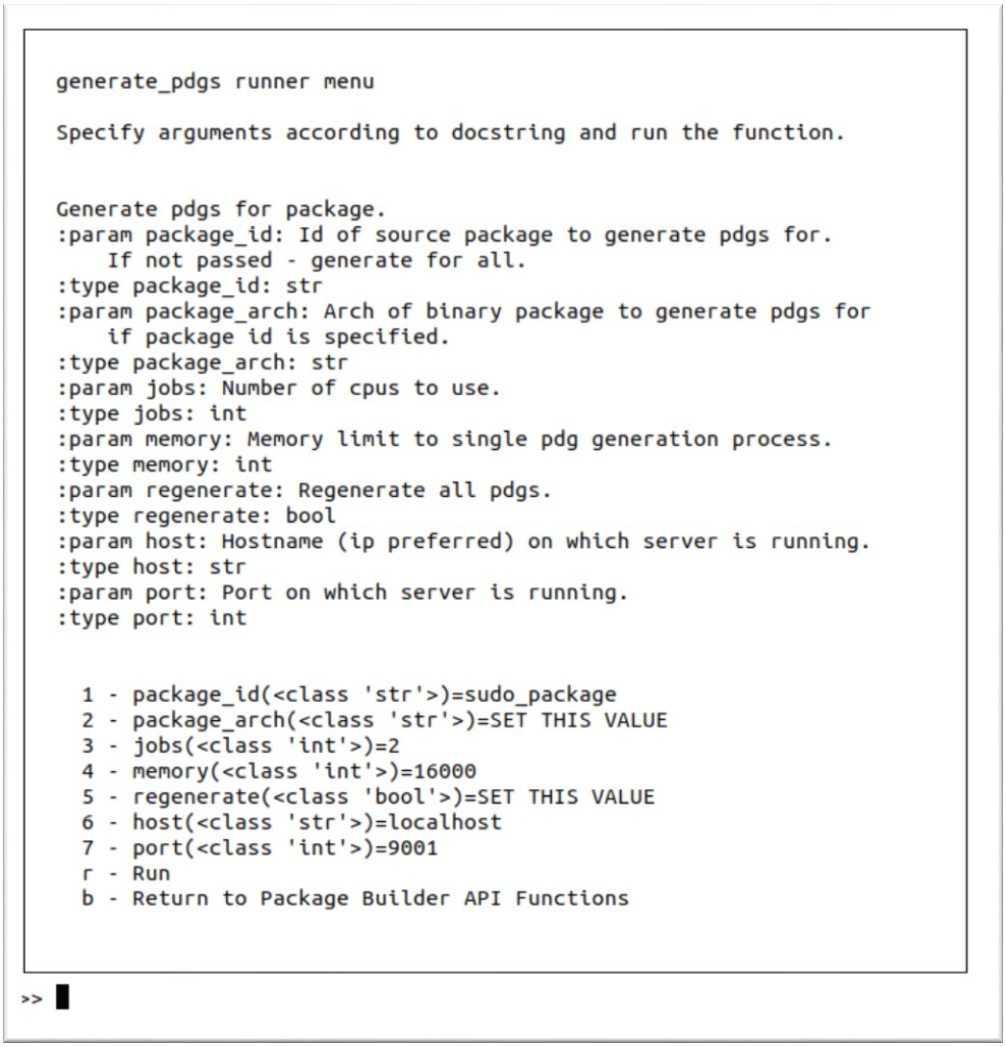


Рисунок 3.15 –Generate PDGs

### 3.6 Запуск анализа компонентов целевого проекта

Для анализа целевого проекта необходимо вернуться в меню "API" и выбрать опцию 2 - "**Analyze Project**" (см. Рисунок 3.6). После перехода отобразится меню "**Analyze Project** " (см. Рисунок 3.16).

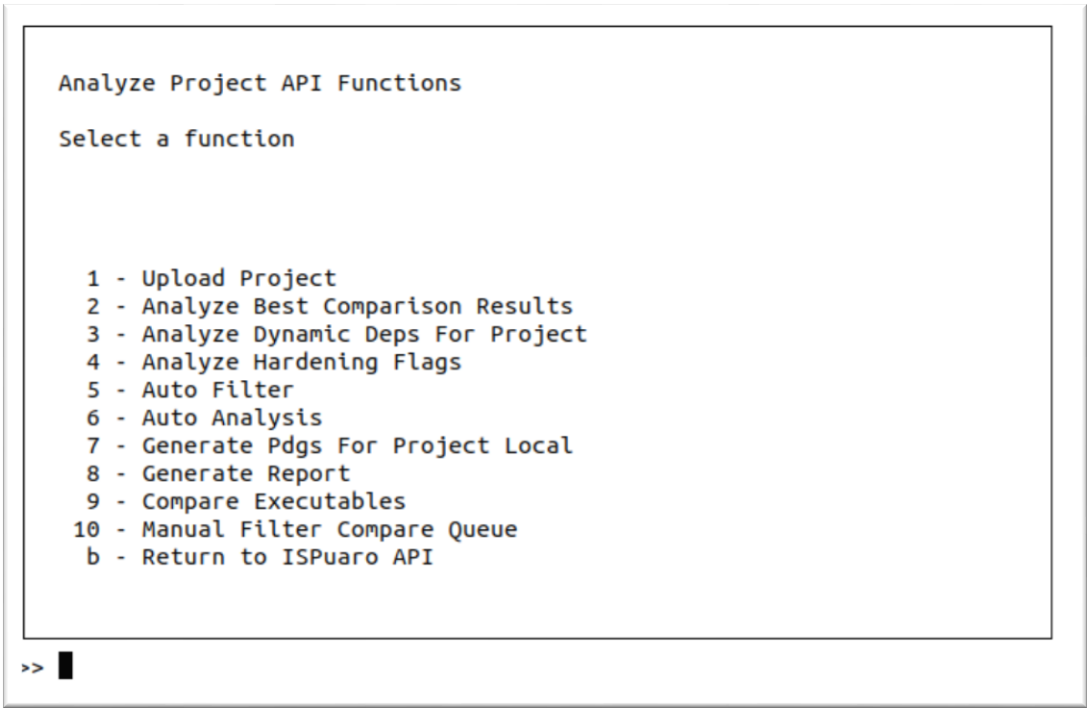


Рисунок 3.16 – Меню Analyze Project

Для запуска автоматизированного анализа компонентов целевого проекта необходимо выбрать опцию 6 - **"Auto Analysis"**. Сам анализ состоит из 7 этапов, которые пользователи могут запускать отдельно.

1. Загрузка проекта, опция 1 - **"Upload Project"**;
2. Загрузка информации для бинарных файлов целевого проекта:
   2.1. Опция 3 - **"Analyze Dynamic Deps For Project"** для сбора информации динамических зависимостей;
   2.2. Опция 4 - **"Analyze Hardening Flags"** для сбора информации о флагах защиты;
3. Задать режим фильтрации для сравнения целевого проекта с базой знаний. В результате фильтрации формируется набор пар для сравнения. Существует два режима:
   3.1. Опция 5 - **"Auto Filter"** для автоматической фильтрации;
   3.2. Опция 10 - **"Manual Filter Compare Queue"** для ручной настройки фильтрации пользователем (пользователь самостоятельно формирует пары для сравнения);
4. Опция 7 - **"Generate PDGs For Project Local"** генерация PDG для целевого проекта;

5. Опция 9 - **"Compare Executables"** для проведения сравнения целевых компонентов программы и проектов в базе знаний;
6. Опция 2 - **"Analyze Best Comparison Results"** сбор информации о проценте вхождения проектов из базы знаний в целевой проект;
7. Опция 8 - **"Generate Report"** для генерации репорта в виде SBOM.

При выборе опции 4 - **"Auto Analysis"** отображается окно запуска автоматизированного анализа с параметрами, которые можно настроить (см. Рисунок 3.17).

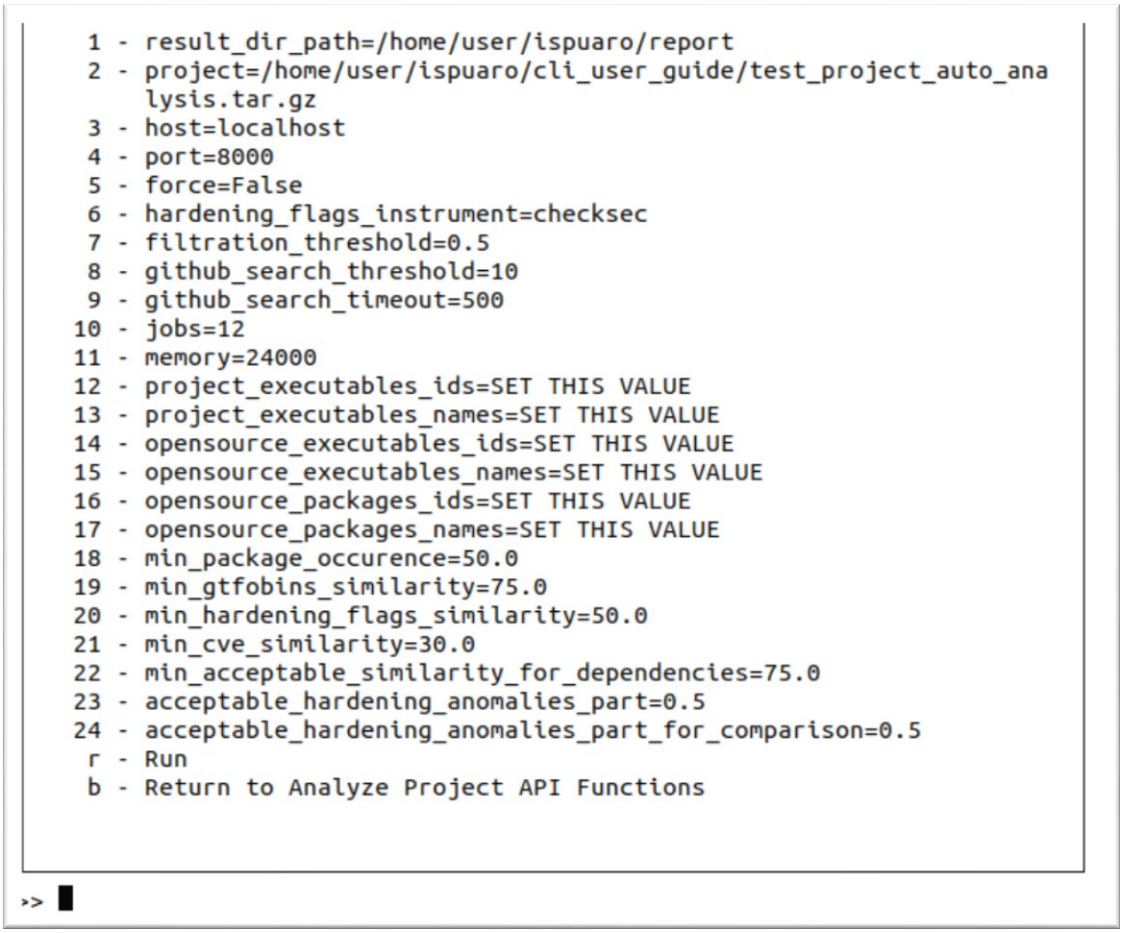


Рисунок 3.17 – Параметры автоматизированного анализа

В параметре 1 - **"project"** необходимо указать путь до целевого проекта, он также должен быть представлен в виде tar-архива. В параметре 9 - **"jobs"** можно указать количество процессов, которые будут задействованы для анализа компонентов. После настройки параметров необходимо запустить анализ с помощью

опции r - **"Run"**. Перед этим необходимо проверить наличие директории report (она должна быть пустой).

После завершения анализа будет выведено сообщение, представленное на Рисунках 3.18, 3.19.


Рисунок 3.18 – Сообщение об успешном завершении анализа

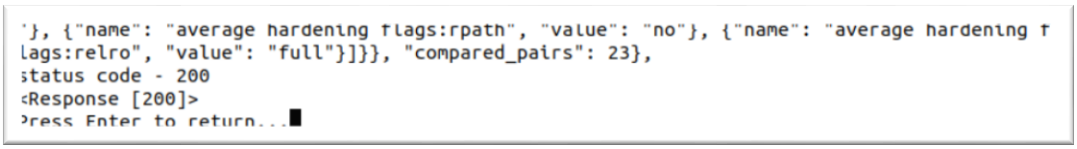


Рисунок 3.19 – Завершение анализа

### 3.7 Результаты анализа компонентов

После завершения анализа результаты будут находиться в директории, которая была указана в параметре **"ARTIFACTS_HOST_PATH"** (см. Рисунок 3.3) перед развертыванием сервера «ISPuaro» (путь в файловой системе, где хранятся загруженные пакеты Debian, проекты для анализа и результаты анализа) и в директории /home/user/ispuaro/report. В данном примере была установлена директория /home/user/artifacts.

```
#Переходим в директорию
cd \
/home/user/artifacts/projects_in_analysis/project_[ID]/analysis_results/reports/
```

Отчет об инвентаризации целевого ПО находится в файле SBOM-[ID].json. В нем представлена информация о найденных компонентах, их зависимостях, а также дефектах, если они присутствуют. На Рисунке 3.20 представлен фрагмент отчёта, в котором указано, что в целевом проекте обнаружен компонент — динамическая библиотека libssl.so.1.1.


Рисунок 3.20 – Обнаружение библиотеки libssl.so.1.1 в целевом проекте

После перечисления всех компонентов в отчёте приводится информация о выявленных дефектах в целевом проекте. Пример описания найденного дефекта изображен на Рисунке 3.21.


```
{
    "id": "CVE-2021-23841",
    "bom-ref": "9f5f0910-f608-48d8-a202-00a614899d34",
    "description": "The OpenSSL public API function X509_issuer_and_serial_hash() attempts to
create a unique hash value based on the issuer and serial number data contained within an X509 certi
ficate. However it fails to correctly handle any errors that may occur while parsing the issuer field
(which might occur if the issuer field is maliciously constructed). This may subsequently result in
 NULL pointer deref and a crash leading to a potential denial of service attack. The function X509_i
ssuer_and_serial_hash() is never directly called by OpenSSL itself so applications are only vulnerabl
 if they use this function directly and they use it on certificates that may have been obtained from
untrusted sources. OpenSSL versions 1.1.1i and below are affected by this issue. Users of these vers
ions should upgrade to OpenSSL 1.1.1j. OpenSSL versions 1.0.2x and below are affected by this issue.
However OpenSSL 1.0.2 is out of support and no longer receiving public updates. Premium support custo
mers of OpenSSL 1.0.2 should upgrade to 1.0.2y. Other users should upgrade to 1.1.1j. Fixed in OpenSS
 1.1.1j (Affected 1.1.1-1.1.1i). Fixed in OpenSSL 1.0.2y (Affected 1.0.2-1.0.2x).",
    "affects": [
        {
            "ref": "788b68f8-9ef8-484f-ac87-58f679d655f8"
        },
        {
            "ref": "c66d9bb8-d742-4a0e-a3e3-d834b550723d"
        }
    ],
```


Рисунок 3.21 – Описание дефекта

Выделенный ID в строке показывает связь найденного дефекта с ранее определенной библиотекой libssl.so.1.1. Этот идентификатор генерируется автоматически и может варьироваться в разных случаях.

Пример визуализации информации о дефектах в базе знаний представлен на Рисунке 3.22.

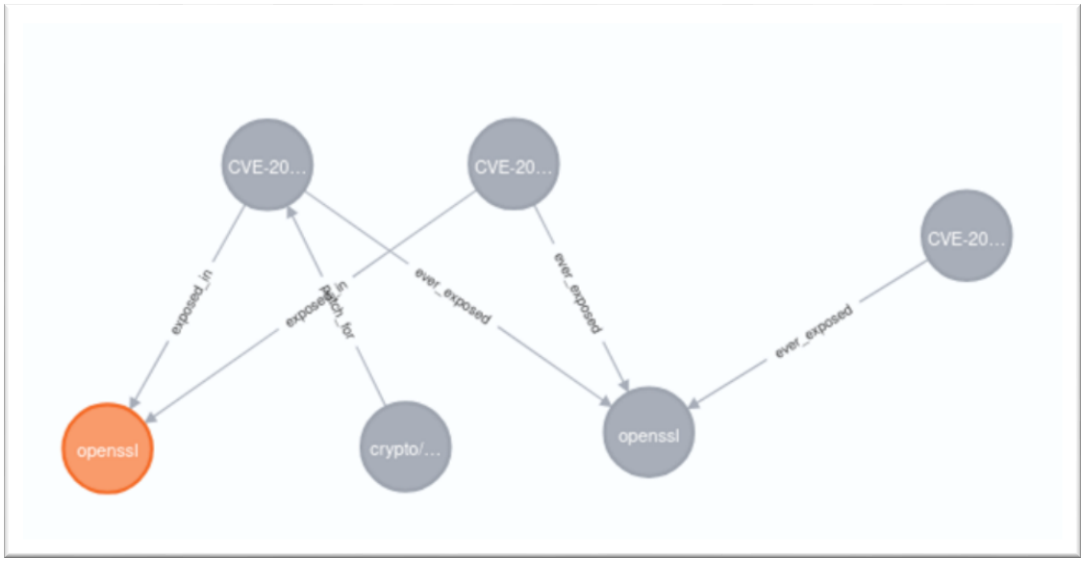


Рисунок 3.22 – Визуализация информации о дефектах в базе знаний

Визуализация результатов анализа представлена на Рисунке 3.23. На нём видно, что такие проекты, как bash, openssl и sudo, были успешно определены с помощью Инструмента «ISPuaro». При наведении курсора на рёбра графа отображается дополнительная информация о степени схожести соответствующих исполняемых файлов.

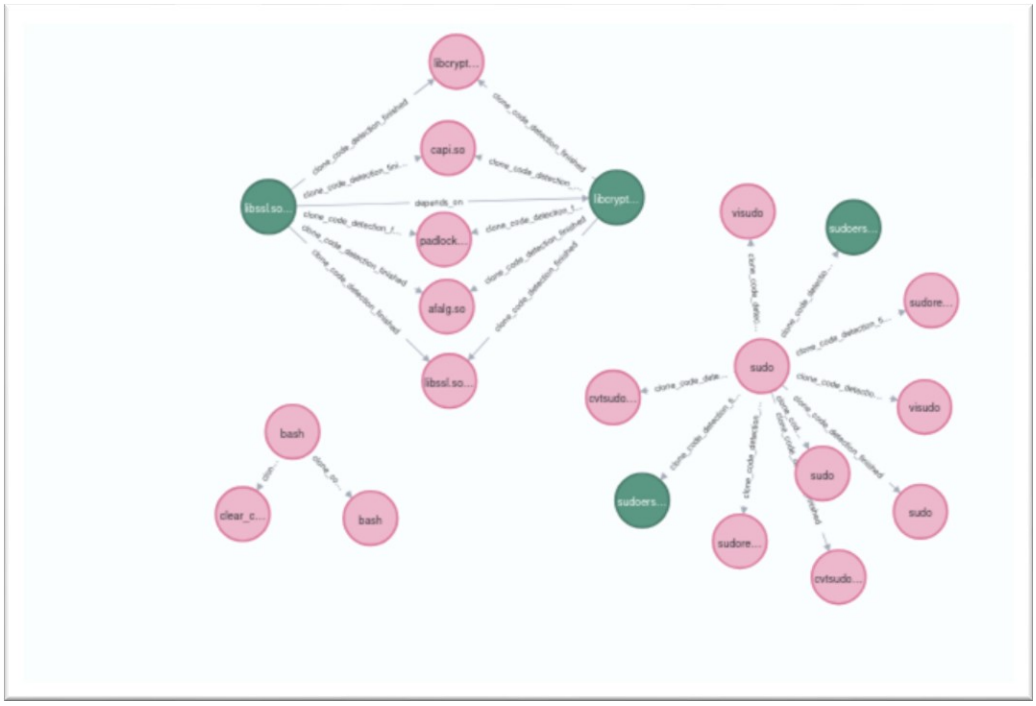


Рисунок 3.23 – Визуализация результатов анализа

### 3.8 Проведение очистки базы знаний и артефактов в файловой системе

Для очистки базы знаний и артефактов в файловой системе необходимо перейти в текстовое меню Dev (см. Рисунок 3.7) и выбрать опцию 4 – **"Clear Database And Artifacts"**. После перехода отобразится меню **"clear_database_and_artifacts"** (см. Рисунок 3.24), в котором необходимо указать параметры **"host", "builder_port"** и **"port"** и выбрать опцию r - **"Run"**.

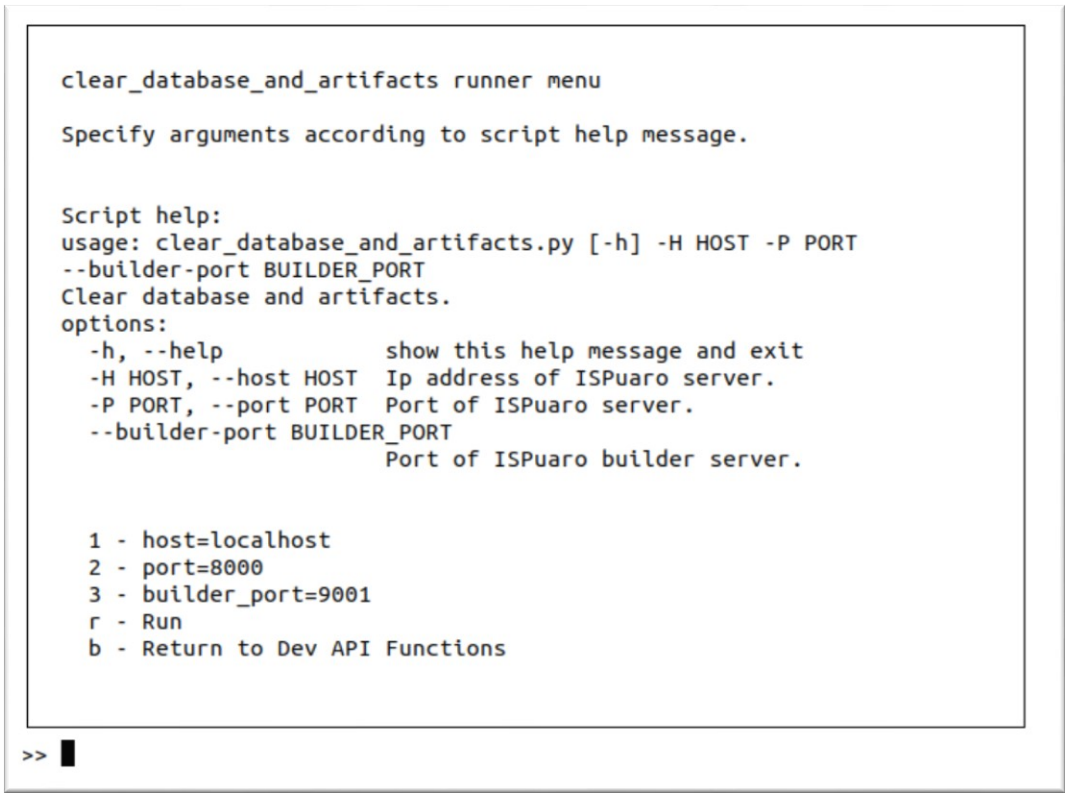


Рисунок 3.24 – Очистка базы знаний и артефактов

# 4. Описание функций системы анализа Инструмента «ISPuaro»

Основными функциями Инструмента «ISPuaro» являются:

- Загрузка проектов (в частном случае пакетов из различных дистрибутивов из открытого доступа) и дефектов в базу знаний, их обработка и составление связей между ними;
- Анализ компонентов целевого проекта и поиск клонов кода;
- Инвентаризация ПО, генерация информации о дефектах.

Реализацию данных функций обеспечивают пять основных модулей Инструмента «ISPuaro»:

1. Модуль загрузки и обработки проектов
2. Модуль загрузки и обработки дефектов
3. Модуль анализа и поиска клонов
4. Модуль генерации отчетов
5. Модуль пользовательского интерфейса

## 4.1 Модуль загрузки и обработки проектов

Данный модуль позволяет загружать, обрабатывать, собирать (build) и хранить проекты, например, пакеты Debian. В проекте может содержаться как исходный код, так и бинарный, а также метаинформация о проекте. После его обработки и сборки создаются связи между проектами, их версиями и зависимостями в базе знаний.

Для более точного поиска интересующих компонентов в Инструменте «ISPuaro» реализована возможность добавлять пользовательские проекты, которые затем могут использоваться для поиска клонов.

## 4.2 Модуль загрузки и обработки дефектов

В Инструменте «ISPuaro» присутствует начальная база знаний загруженных и обработанных дефектов (идентификаторы дефектов, а также дефекты, реализующие механизмы контроля доступа и привилегий) из открытых источников. При обработке данных о дефекте ему ставится в соответствие информация о производителе, дефектных компонентах (при их наличии), а также о ПО и его версиях, в которых этот дефект проявляется. Есть возможность автоматизированно обновлять базу знаний новыми дефектами, если в системе, где запущен «ISPuaro», есть доступ в сеть Интернет.

Дополнительно реализован функционал добавления информации о пользовательских дефектах в базу знаний.

### 4.3 Модуль анализа и поиска клонов

Данный модуль состоит из нескольких этапов:

1. Анализ компонентов целевого ПО с целью выявления динамических зависимостей и поиска флагов защиты;
2. Генерация промежуточного представления бинарных компонентов ПО для создания PDG;
3. Предварительный анализ, реализующий поиск информации о компонентах, использующий в качестве одного из методов сопоставление бинарных файлов с исходным кодом, для ускорения этапа поиска клонов кода в базе знаний путём уменьшения числа сравниваемых пар элементов базы и целевого ПО;
4. Поиск клонов кода на основе полученных пар сравнения после предварительного анализа. Данный этап требует много ресурсов, но позволяет точнее определять компоненты целевого ПО, в частности их версии;
5. Перенос информации о дефектах из базы знаний в пользовательский проект при помощи результатов поиска клонов кода.

### 4.4 Модуль генерации отчетов

Данный модуль генерирует итоговый отчет об анализе компонентов целевого ПО на основе результатов работы всех этапов и полученной информации из базы знаний. Отчет представлен в стандартизированном формате SBOM (CycloneDX). В нем содержится вся основная информация: компоненты, их зависимости, версии, дефекты и другая метаинформация.

### 4.5 Модуль пользовательского интерфейса

Данный модуль реализует пользовательский интерфейс «ISPuaro», предоставляющий доступ ко всему функционалу инструмента через текстовое меню. Интерфейс также поддерживает возможность развертывания сервера Инструмента «ISPuaro».

# 5. Описание процессов, обеспечивающих поддержание жизненного цикла ПО

## 5.1 Процессы разработки и совершенствования ПО

Разработка Инструмента «ISPuaro», реализующего анализ компонентов и сопровождающих артефактов ПО, ведется по методологии Agile с привлечением современных средств повышения качества кода.

1. Для хранения кода используется система контроля версий git, и изменения в основной ветке должны пройти все тесты, а также инспекцию кода (code review) другими разработчиками.
2. При разработке используется практика непрерывной интеграции (continuous integration): при помощи сервера тестирования Jenkins происходят регулярные автоматические сборки с последующим запуском тестов, упомянутых в предыдущем пункте.
3. При написании кода разработчики должны придерживаться строгих правил оформления кода. Нарушение этих правил не позволит пройти этап инспекции кода.

## 5.2 Поддержка пользователей ПО

Развертывание дистрибутива Инструмента «ISPuaro» на рабочей станции не требует настройки ОС, установки внешнего ПО или каких-либо других дополнительных действий. Если есть доступ в сеть Интернет, то можно обновлять информацию о новых дефектах.

Со своей стороны, ИСП РАН постоянно совершенствует Инструмент «ISPuaro», применяя в жизненном цикле разработки передовые методики. Добавление новых и улучшение существующих алгоритмов ведется в инициативном порядке. Обновления Инструмента «ISPuaro» передаются пользователям через согласованные с ними каналы распространения обновлений.

## 5.3 Необходимый персонал для разработки и поддержки

Для разработки и поддержки программного продукта необходима соответствующая квалификация разработчиков. Это вызвано следующими причинами:

1. Специфическая предметная область, требующая глубоких знаний одновременно в нескольких областях: статический анализ кода,

динамический анализ кода, компиляторные технологии, базы знаний, декомпиляция, реверс-инжиниринг.

2. Требования к производительности системы, из-за чего необходимо применять эффективные алгоритмы, в т. ч. хорошо масштабируемые по нескольким вычислительным ядрам.

В силу приведенных причин коллектив разработчиков Инструмента «ISPuaro» формируется из специалистов, получивших высшее профильное образование.

Для гарантийного обслуживания задействовано четыре научных сотрудника, для Технической поддержки задействован один научный сотрудник, для модернизации программного обеспечения задействованы три научных сотрудника.

Адрес электронной почты, по которому можно обратиться по вопросам, связанным с Инструментом «ISPuaro» — ispuaro@ispras.ru.

## 6. Перечень терминов и сокращений

| | |
|---|---|
| Система | Инструмент «ISPuaro» |
| TUI | Текстовый интерфейс |
| CLI | Интерфейс командной строки |
| PDG | Граф зависимостей программы |
| WEB | Веб-интерфейс |
| ID | Уникальный идентификатор |
| SBOM | Реестр компонентов программного обеспечения |