Лабораторная работа №2
Статический анализ кода программ

Цель работы

Получить навыки статического анализа кода программ с помощью статических анализаторов PVS-Studio и cppcheck

Работа со статическим анализатором cppcheck

Статический анализатор cppcheck является продуктом с открытым исходным кодом и входит в состав программного обеспечения основных дистрибутивов ОС Linux. Данный анализатор кроме варианта с командной строкой также имеет и графическую оболочку, однако в рамках лабораторной работы будет рассматриваться работа только с вариантом командной строки.

Рассмотрим основные параметры, используемые при запуске анализатора.

Параметр --std=<id> позволяет указать стандарт языков Си и Си++, использованных в проверяемом коде. Допустимы значения для <id>: c89, c99, c11, c++03, c++11, c++14, c++17, c++20

Параметр --enable=<id> позволяет указать глубину анализа. Если данный параметр не указан, то отчете указываются только критические ошибки. Допустимы следующие значения <id>:

- **all** включает все проверки. Рекомендуется только, если анализируются все файлы проекта;
- **warning** включает предупреждения;
- **style** включает проверку стиля кодирования. В отчете будут сообщения с метками 'style', 'performance' и 'portability';
- **performance** включает предупреждения по снижению быстродействия программы;
- **portability** включает предупреждения с проблемами портирования программы на другие платформы;
- **information** включает информационные сообщения;
- **unusedFunction** включает предупреждения о неиспользованных функциях. Актуально только при проверке всех файлов проекта одновременно. Не работает при много поточной проверке;
- **missingInclude** включает предупреждения, если отсутствуют указанные в директиве #include файлы.

Если надо включить несколько дополнительных проверок, то они указываются через запятую без пробелов, например: --enable=style,unusedFunction

Параметр -I <dir> позволяет указать каталог, где находятся заголовочные файлы, если путь не указан в директиве #include.

Отчет по умолчанию формируется в текстовом виде. Для формирования отчета в формате XML используется параметр --xml. При этом отчет выводится в поток сообщений об ошибках, поэтому для его перенаправления в файл необходимо указывать номер потока 2.

При проверке можно указать как отдельный файл, так и каталог. При указании каталога в нем проверяются все файлы с исходным кодом (для Си++ с суффиксом .cpp). Рассмотрим несколько примеров запуска проверки.

Запуск проверки всех файлов в каталоге prj1 с выдачей ошибок, предупреждений и поиском неиспользуемых функций для кода, соответствующего стандарту c++14:

```
cppcheck --std=c++14 --enable=warning,unusedFunction  prj1
```

Запуск с параметрами, аналогичными предыдущему случаю, но с сохранением результата в текстовый файл result.txt:

```
cppcheck --std=c++14 --enable=warning,unusedFunction  prj1 2> result.txt
```

Однако следует учитывать, что в текстовом файле будут присутствовать коды для «цветного» вывода. Для получения в файле чистого текста следует использовать следующую команду:

```
cppcheck --std=c++14 -q prj1 2>&1 | sed 's/\x1b\[[0-9;]*m//g' > result.txt
```

Здесь параметр -q подавляет вывод сообщений о прогрессе в стандартный поток вывода, а команда sed отфильтровывает коды для «расцвечивания» отчета.

В cppcheck возможно получение отчета в html-виде для просмотра в браузере. Для этого необходимо выполнить следующие действия:

1. С помощью cppcheck провести анализ и сохранить результат в XML-файл
2. Создать каталог для html-отчета
3. С помощью cppcheck-htmlreport создать html-отчет
4. Открыть в браузере индексный файл html-отчета

Первое действие можно выполнить например так, сохранив результат в файл report.xml:

```
cppcheck --std=c++14 --enable=warning --xml  prj1  2> report.xml
```

Тогда третье действие можно выполнить таким образом, сохранив отчет в каталог report:

```
cppcheck-htmlreport --file=report.xml --report-dir report
```

Подготовка проекта для работы с PVS-Studio

Для работы с PVS-Studio проект должен быть специально подготовлен. При этом важную роль играет то, какие программные средства используются для компиляции и сборки проекта. Кроме того, PVS-Studio является коммерческим продуктом и для его работы в большинстве случаев требуется покупка лицензии. Однако, возможно бесплатное использование этого продукта для некоторых типов проектов, в том числе и академических. Для этого в каждом файле исходного кода должен быть помещен комментарий следующего содержания:

```
// This is a personal academic project. Dear PVS-Studio, please check it.
// PVS-Studio Static Code Analyzer for C, C++, C#, and Java: https://pvs-studio.com
```

Кроме этого необходимо сгенерировать лицензионный ключ для свободного использования. Это делается командой

```
pvs-studio-analyzer credentials PVS-Studio Free FREE-FREE-FREE-FREE
```

После генерации будет выдано сообщение о времени действия лицензионного ключа, а также полное имя файла ключа. Как правило, ключ помещается в каталог, используемый анализатором по умолчанию и при запуске путь к нему указывать не надо. Если же ключ будет перемещен, то следует при старте анализатора использовать параметр

```
--lic-file  путь_к_файлу_лицензии
```

Таким образом, студенты для проверки своих проектов могут пользоваться PVS-Studio бесплатно.

Рассмотрим самый простой вариант сборки с помощью утилиты make. Для анализа кода надо выполнить три действия:

- включить во все файлы исходного кода комментарий для использования анализатора без лицензии;
- создать файл конфигурации PVS-Studio.cfg;
- модифицировать Makefile, так, чтобы в процессе компиляции использовался статический анализатор.

Подробно о настройках проектов сказано в онлайн-руководстве https://pvs-studio.ru/ru/docs/manual/0036/ . Кроме того, можно воспользоваться примером интеграции PVS-Studio в проект со сборкой утилитой make https://github.com/viva64/pvs-studio-makefile-examples.git. Преимущество готового примера заключается в том, что его можно применить к своему проекту лишь слегка изменив файл Makefile.

Проверка проекта «Аутентификация с использованием USB-накопителя»

Код проекта находится в открытом репозитории https://gitlab.com/winwood/usb-auth. Репозиторий содержит программные модули проекта, а также makefile для сборки.

Данный проект состоит из 6 программных модулей. Однако, все они компилируются одновременно, так как код модулей включается в главный модуль директивой *include* препроцессора.

Главный программный модуль, pam_usb.cpp, является начальной точкой исполнения разработанного модуля аутентификации USB–накопителя. Здесь расположен код основного модуля аутентификации «`auth`» и модуля проверки учетной записи «`account`» системы подключаемых модулей аутентификации (PAM).

Программный модуль Authentication.cpp является реализацией модулей «auth» и «account», а именно — получение и проверка списка подключенных устройств в системе, проверка серийного номера устройства и имени учетной записи, а также проверка срока действия учетной записи пользователя системы.

Программный модуль VerifierSchemes.cpp содержит реализацию проверки наличия открытого ключа криптографического алгоритма в системе и подтверждения данных аутентификации с устройства с помощью цифровой подписи.

Программный модуль DateTime.cpp производит чтение текущей даты и времени системы, а также расчет окончания срока действия учетной записи на основе полученных значений из данных аутентификации.

Программный модуль Controls.cpp производит управление подключенными устройствами в системе, а именно проверку доступности подключенного устройства, фильтрацию мониторинга подключаемых устройств и выгрузку с устройства данных аутентификации, подписанных цифровой подписью.

Программный модуль Data.cpp реализует временное хранение полученных данных аутентификации с подключенного устройства.

Перед сборкой проекта необходимо установить следующие пакеты программных библиотек:

Список пакетов дается для операционной системы Debian Linux. Перед проверкой проекта статическим анализатором необходимо убедится в том, что проект успешно собирается в обычной конфигурации. Результатом сборки является модуль аутентификации pam_usb.so в формате динамической библиотеки.

Для статического анализа проекта выполнить все действия по подготовке к анализу: добавить комментарий об академическом проекте, создать конфигурационный файл, модифицировать makefile.

Работа с git через прокси-сервер

Если работы выполняются во внутренней сети организации, в которой выход в сеть Интернет осуществляется через прокси-сервер, то работать с репозиториями напрямую не получится. Однако, в git есть возможность осуществлять взаимодействие с репозиториями по протоколам http и https через прокси. Для этого необходимо в настройках добавить параметр **http.proxy**. Это делается командой git config следующим образом

```
git config --global|--local http.proxy <PROXY_URL>
```

Выбирая между параметрами --global или --local можно указать, будут ли эти настройки работать для всех репозиториев или только для одного конкретного. Параметр <PROXY_URL> — адрес прокси в формате **protocol://proxyhost:port**. Где proxyhost — это DNS-имя прокси-сервера или его IP-адрес. Например **http://rokot.ibst.psu:3128 —** адрес прокси в сети кафедры ИБСТ (на начало 2022г.).

В зависимости от настроек, прокси может требовать, а может и не требовать авторизацию. Если требуется авторизация, то самый простой способ — добавить имя пользователя в <PROXY_URL>. Тогда он будет выглядеть так: **protocol://user@|proxyhost:port**. В этом случае, каждый раз при установке соединения через прокси будет запрашиваться пароль.

Задание к лабораторной работе

1. Клонировать репозиторий https://github.com/viva64/pvs-studio-makefile-examples.git, выполнить сборку и статический анализ проекта из папки example-1 с использованием PVS-Studio.
2. Клонировать репозиторий https://gitlab.com/winwood/rbpo.git
3. Провести статический анализ кода для проекта из папки prj1 с помощью cppcheck.
4. Выполнить подготовку к анализу, провести статический анализ кода для проекта из папки prj1 с использованием PVS-Studio.
5. Сравнить результаты анализа двух анализаторов для проекта prj1. Описать выявленные ошибки и исправить их.
6. Провести статический анализ кода для проекта из папки prj2 с помощью cppcheck.
7. Выполнить подготовку к анализу, провести статический анализ кода для проекта из папки prj2 с использованием PVS-Studio.

8. Сравнить результаты анализа двух анализаторов для проекта prj2. Описать выявленные ошибки и исправить их.
9. Провести статический анализ кода для проекта из папки prj3 с помощью cppcheck.
10. Выполнить подготовку к анализу, провести статический анализ кода для проекта из папки prj3 с использованием PVS-Studio.
11. Сравнить результаты анализа двух анализаторов для проекта prj3. Описать выявленные ошибки и исправить их.
12. Клонировать репозиторий https://gitlab.com/winwood/usb-auth проекта «Аутентификация с использованием USB-накопителя»
13. Провести статический анализ кода для проекта «Аутентификация с использованием USB-накопителя» с помощью cppcheck.
14. Подготовить проект «Аутентификация с использованием USB-накопителя» к проверке анализатором PVS-Studio
15. Выполнить статический анализ проекта «Аутентификация с использованием USB-накопителя» с использованием PVS-Studio/
16. Сравнить результаты анализа двух анализаторов для проекта «Аутентификация с использованием USB-накопителя». Описать выявленные ошибки и исправить их при необходимости.