

Извештај примене алата за верификацију над библиотеком за детекцију лица

Марија Ерић
marija.eric@matf.bg.ac.rs

Јануар 2023

Сажетак

Пројекат на коме је извршена анализа је библиотека за детекцију лица, написана у $C++$. Библиотека је отвореног кода и може се пронаћи на наредном линку. Примена алата је извршена на главној грани, над комитом чији је хеш код: *ec528ce43af9de94bf2fab308ce2d6270584881c*. У коду нису пронађени већи пропусти, поред некоришћених променљивих и цурења меморије.

Садржај

1	Верификација софтвера	2
1.1	Динамичка анализа	2
1.1.1	Измена програма	2
1.1.2	<i>Gcov</i>	2
1.2	Профајлирање	5
1.2.1	<i>Memcheck</i>	5
1.2.2	<i>Massif</i>	6
1.3	Статичка анализа	9
1.3.1	<i>CppCheck</i>	9
2	Закључак	12
3	Покретање скрипти за репродуковање резултата	12
3.1	Покретање алата	13

1 Верификација софтвера

1.1 Динамичка анализа

1.1.1 Измена програма

Да бисмо могли да репордукујемо резултате добијене приликом извршавања програма, неопходна је измена у оквиру *detect – camera*. На слици 1 је приказана измена. На левој слици се налази оригиналан код, а на десној измењени код.

<pre>67 VideoCapture cap; 68 Mat im; 69 70 if(isdigit(argv[1][0])) 71 { 72 cap.open(argv[1][0]-'0'); 73 if(! cap.isOpened()) 74 { 75 cerr << "Cannot open the camera." << endl; 76 return 0; 77 } 78 }</pre>	<pre>67 VideoCapture cap; 68 Mat im; 69 70 if(isdigit(argv[1][0])) 71 { 72 cap.open("../TestSamples/camera-test.mp4"); 73 if(! cap.isOpened()) 74 { 75 cerr << "Cannot open the camera." << endl; 76 return 0; 77 } 78 } 79</pre>
---	--

Слика 1: Измене у оквиру *detect – camera*

1.1.2 *Gcov*

Gcov алат за одређивање покривености кода приликом извршавања програма (*engl.codecoverage*). Користи се заједно са *gcc* компајлером да би се анализирао програм и утврдило како се може креирати ефикаснији, бржи код и да би се тестовима покрили делови програма. Зарад лепше репрезентације резултата детекције покривености кода извршавањем тест примера, користи се алат *lcov* [1].

Начин покретања алата: Приликом компилације неопходно је користити додатне опције компајлера које омогућавају снимање колико је пута која линија, грана и функција извршена. Детаљи се налазе на дну текста. У наставку се налази покретање алата *lcov* и генерисање *html* извештаја.

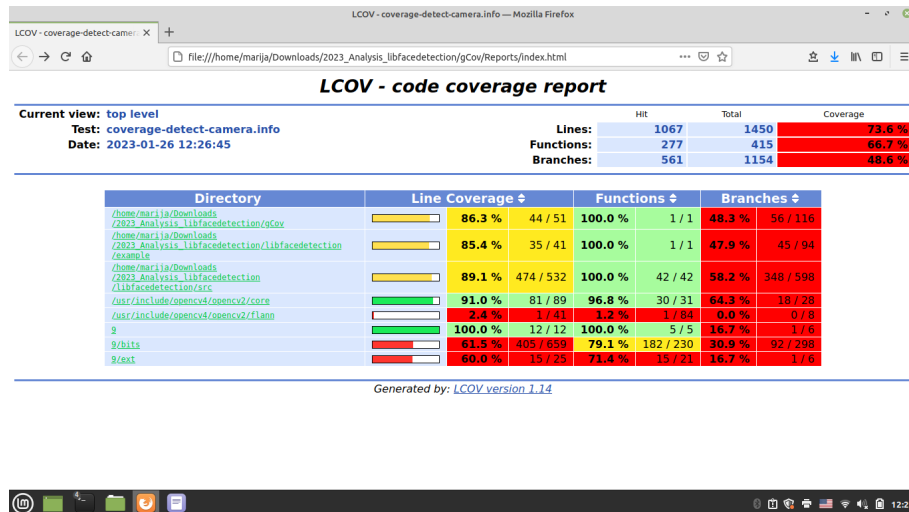
```
lcov --rc lcov_branch_coverage=1 -c -d ../TestSamples/ -o coverage-detect.info
```

```
genhtml --rc lcov_branch_coverage=1 -o Reports coverage-detect.info
```

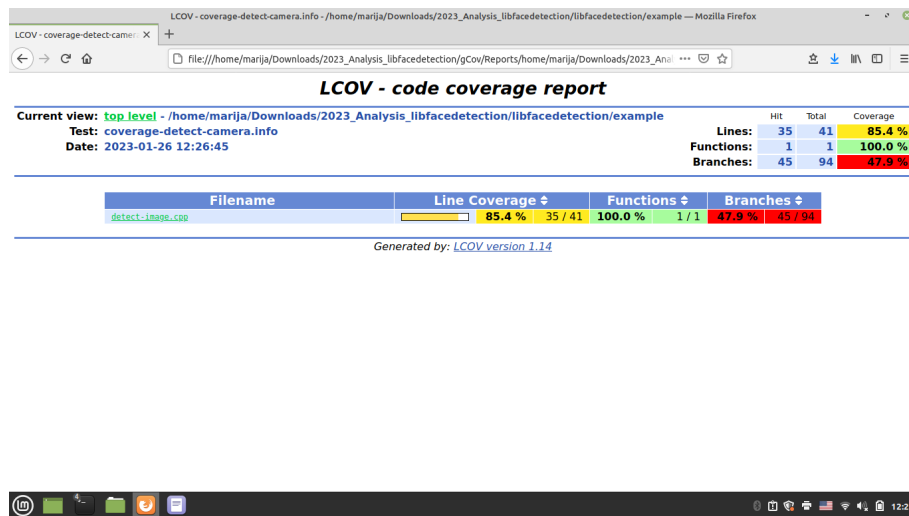
```
firefox Reports/index.html
```

На слици 2 се налази извештај за оба примера употребе библиотеке: *detect – image* и *detect – camera*. Анализирано је исправно покретање програма. Као што видимо покривеност кода је велика. Све функције дефинисане у оквиру библиотеке су искоришћене. Покривеност грана је мања (око 50%). У оквиру *detect – image* и *detect – camera* је велика покривеност

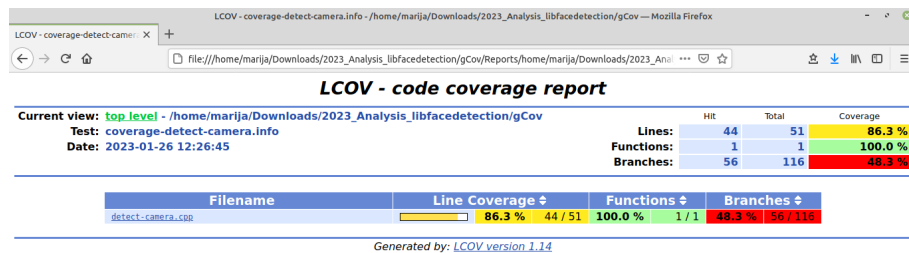
наредби. Једине неизвршене су наредбе обраде погрешног улаза. Детаљан извештај се може видети на 3 и 4.



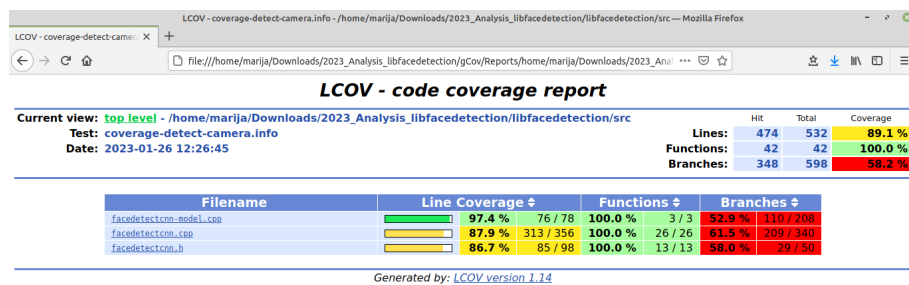
Слика 2: Извештај о покривеност кода



Слика 3: Покривеност кода *detect – image*



Слика 4: Покривеност кода *detect – camera*



Слика 5: Покривеност кода библиотеке

Резултати: Мала покривеност је у оквиру *opencv* библиотеке, јер се користи само једна функција дефинисана у овој библиотеци, као и функције из *math* библиотеке.

Дакле, након покретања алата *gscov* можемо да закључимо да имамо велику покривеност кода, као и да немамо некоришћене функције у оквиру библиотеке за детекцију лица.

1.2 Профајлирање

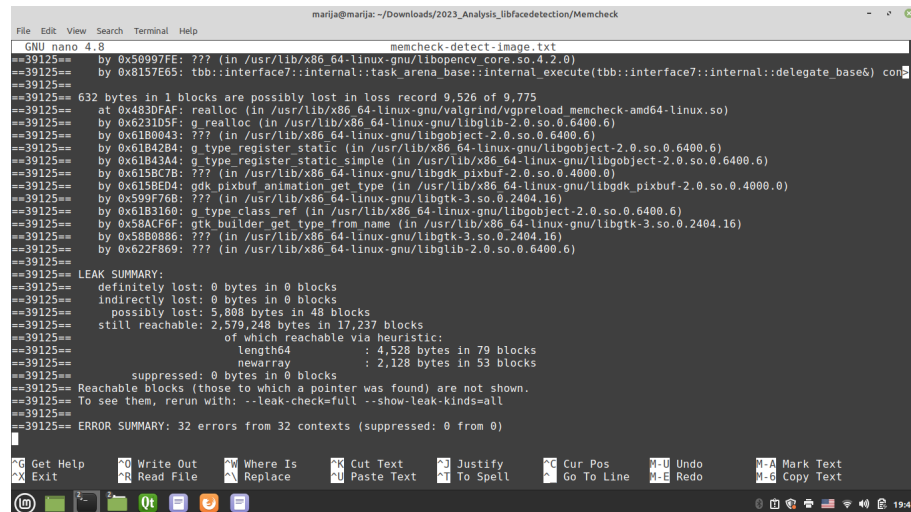
Са обзиром на то да се у оквиру библиотеке користи динамичка алокација меморије, битно је да испитамо да ли је дошло до цурења меморије. Из тог разлога вршимо профајлирање програма *detect – image* и *detect – camera*. У оквиру анализе су коришћени алати *Memcheck* и *Massif*.

1.2.1 Memcheck

Начин покретања: Битно је да превођење извршимо у *debug* моду. Након тога, покрећемо *Memcheck*, део извештаја је приказан на сликама 6 и 7, док се целокупан излаз може наћи у оквиру фолдера *Memcheck* на *github* репозиторијуму.

```
valgrind -s --leak-check=full --log-file="memcheck-detect-image.txt"
../TestSamples/detect-image ../TestSamples/image-test.jpg
```

```
valgrind -s --leak-check=full --log-file="memcheck-detect-camera.txt"
../TestSamples/detect-camera 0
```



```
GNU nano 4.8 memcheck-detect-image.txt
==39125== by 0x50997FE: ??? (in /usr/lib/x86_64-linux-gnu/libopencv_core.so.4.2.0)
==39125== by 0x8157E65: tbb::interface7::internal::task_arena_base::internal_execute(tbb::interface7::internal::delegate_base6) const
==39125==
==39125== 632 bytes in 1 blocks are possibly lost in loss record 9,526 of 9,775
==39125== at 0x483DFAF: realloc (in /usr/lib/x86_64-linux-gnu/valgrind/vgpreload_memcheck-amd64-linux.so)
==39125== by 0x6231D5F: g_realloc (in /usr/lib/x86_64-linux-gnu/libglib-2.0.so.0.6400.6)
==39125== by 0x6180043: ??? (in /usr/lib/x86_64-linux-gnu/libgobject-2.0.so.0.6400.6)
==39125== by 0x61842B4: g_type_register_static (in /usr/lib/x86_64-linux-gnu/libgobject-2.0.so.0.6400.6)
==39125== by 0x61843A4: g_type_register_static_simple (in /usr/lib/x86_64-linux-gnu/libgobject-2.0.so.0.6400.6)
==39125== by 0x615BC7B: ??? (in /usr/lib/x86_64-linux-gnu/libgdk_pixbuf-2.0.so.0.4000.0)
==39125== by 0x615BED4: gdk_pixbuf_animation_get_type (in /usr/lib/x86_64-linux-gnu/libgdk_pixbuf-2.0.so.0.4000.0)
==39125== by 0x599F768: ??? (in /usr/lib/x86_64-linux-gnu/libgtk-3.so.0.2404.16)
==39125== by 0x6183160: g_type_class_ref (in /usr/lib/x86_64-linux-gnu/libgobject-2.0.so.0.6400.6)
==39125== by 0x58ACF6F: gtk_builder_get_type_from_name (in /usr/lib/x86_64-linux-gnu/libgtk-3.so.0.2404.16)
==39125== by 0x58B0886: ??? (in /usr/lib/x86_64-linux-gnu/libgtk-3.so.0.2404.16)
==39125== by 0x622F869: ??? (in /usr/lib/x86_64-linux-gnu/libglib-2.0.so.0.6400.6)
==39125==
==39125== LEAK SUMMARY:
==39125== definitely lost: 0 bytes in 0 blocks
==39125== indirectly lost: 0 bytes in 0 blocks
==39125== possibly lost: 5,808 bytes in 48 blocks
==39125== still reachable: 2,579,248 bytes in 17,237 blocks
==39125== of which reachable via heuristic:
==39125== length64 : 4,528 bytes in 79 blocks
==39125== newarray : 2,128 bytes in 53 blocks
==39125== suppressed: 0 bytes in 0 blocks
==39125== Reachable blocks (those to which a pointer was found) are not shown.
==39125== To see them, rerun with: --leak-check=full --show-leak-kinds=all
==39125==
==39125== ERROR SUMMARY: 32 errors from 32 contexts (suppressed: 0 from 0)
```

Слика 6: Део излаза алата *Memcheck* над *detect – image*.

```

marja@marja: ~/Downloads/2023_Analysis_libfacedetection/Memcheck
GNU nano 4.8 memcheck-detect-camera.txt
==33263== 640 bytes in 1 blocks are possibly lost in loss record 9,528 of 9,794
==33263== at 0x483DD99: calloc (in /usr/lib/x86_64-linux-gnu/valgrind/vgpreload_memcheck-amd64-linux.so)
==33263== by 0x40149CA: allocate_dtv (dl-tls.c:286)
==33263== by 0x40149CA: dl_allocate_tls (dl-tls.c:532)
==33263== by 0x6394322: allocate_stack (allocatesetask.c:622)
==33263== by 0x6394322: pthread_create@@GLIBC_2.2.5 (pthread_create.c:660)
==33263== by 0x62A102A: ??? (in /usr/lib/x86_64-linux-gnu/libglib-2.0.so.0.6400.6)
==33263== by 0x627DD03: g_thread_new (in /usr/lib/x86_64-linux-gnu/libglib-2.0.so.0.6400.6)
==33263== by 0x847D9A3: ??? (in /usr/lib/x86_64-linux-gnu/libgio-2.0.so.0.6400.6)
==33263== by 0x847D9A3: ??? (in /usr/lib/x86_64-linux-gnu/libgio-2.0.so.0.6400.6)
==33263== by 0x84712F3: g_bus_get_sync (in /usr/lib/x86_64-linux-gnu/libgio-2.0.so.0.6400.6)
==33263== by 0x1002DE4E: ??? (in /usr/lib/x86_64-linux-gnu/gio/modules/libgvfsdbus.so)
==33263== by 0x61DD16C: g_type_create_instance (in /usr/lib/x86_64-linux-gnu/libgobject-2.0.so.0.6400.6)
==33263== by 0x61BC34C: ??? (in /usr/lib/x86_64-linux-gnu/libgobject-2.0.so.0.6400.6)
==33263== by 0x61B0B44: g_object_new_with_properties (in /usr/lib/x86_64-linux-gnu/libgobject-2.0.so.0.6400.6)
==33263==
==33263== LEAK SUMMARY:
==33263== definitely lost: 0 bytes in 0 blocks
==33263== indirectly lost: 0 bytes in 0 blocks
==33263== possibly lost: 9,536 bytes in 47 blocks
==33263== still reachable: 2,041,743 bytes in 19,076 blocks
==33263== of which reachable via heuristic:
==33263== length64 : 4,488 bytes in 78 blocks
==33263== newarray : 2,128 bytes in 53 blocks
==33263== suppressed: 0 bytes in 0 blocks
==33263== Reachable blocks (those to which a pointer was found) are not shown.
==33263== To see them, rerun with: --leak-check=full --show-leak-kinds=all
==33263==
==33263== ERROR SUMMARY: 31 errors from 31 contexts (suppressed: 0 from 0)

```

Слика 7: Део излаза алата *Memcheck* над *detect – camera*.

Добијамо јако сличне излазе за оба програма. На основу сажетка алата видимо да нема цурења меморије. Такође видимо да имамо могући губитак (око 5000 бајтова у оба програма). Када испитамо стек позива видимо да је могући губитак изазвао позив функције *calloc*, али закључујемо да не постоји губитак меморије који је изазван од стране програмера.

1.2.2 *Massif*

Massif представља профајлер *heap* меморије. Овај алат мери колико *heap* меморије програм користи (корисна меморије ђ додатна меморија за администрацију). У оквиру ове анализе је мерена и величина стека коришћена у оквиру програма.

Мотивација иза коришћења и овог алата лежи у томе да Мемчек не може да детектује сва потенцијална цурења меморије.

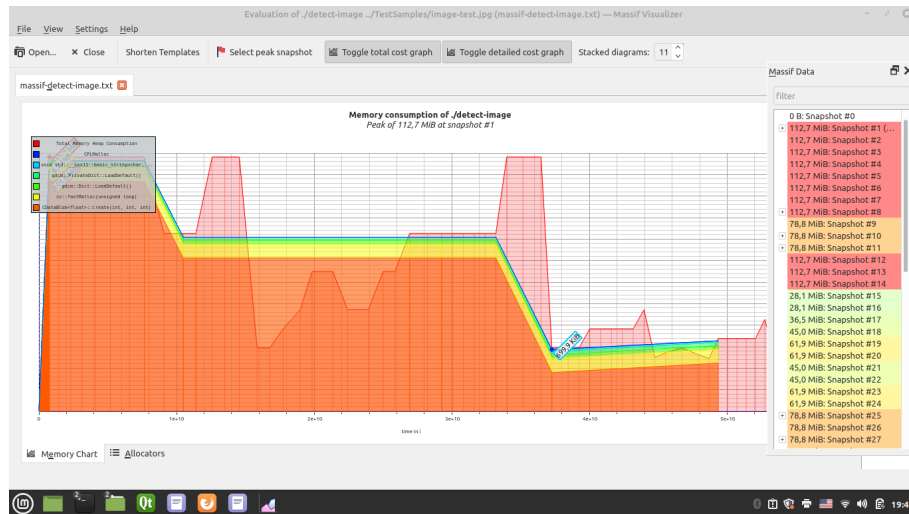
Pokretanje Massifa za detect-camera

```
valgrind --tool=massif --stacks=yes --massif-out-file="massif-detect-camera.txt" ../../TestSa
ms_print massif-detect-camera.txt > massif-detect-camera-msprint.txt
```

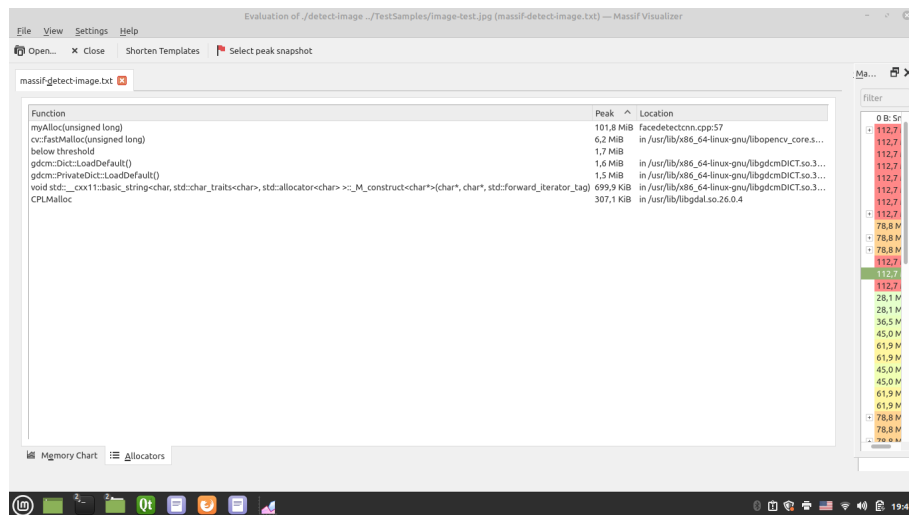
```
valgrind --tool=massif --stacks=yes --massif-out-file="massif-detect-image.txt" ../../TestSa
```

За визуелизацију резултата коришћен *massif – visualizer*, који је преузет са <https://github.com/KDE/massif-visualizer>, где се може пронаћи детаљно упутство за инсталирање и покретање.

На сликама 8 и 9 се налазе извештаји везани за програм *detect – image*, док се на сликама 10 и 11 налазе извештаји везани за програм *detect – camera*.

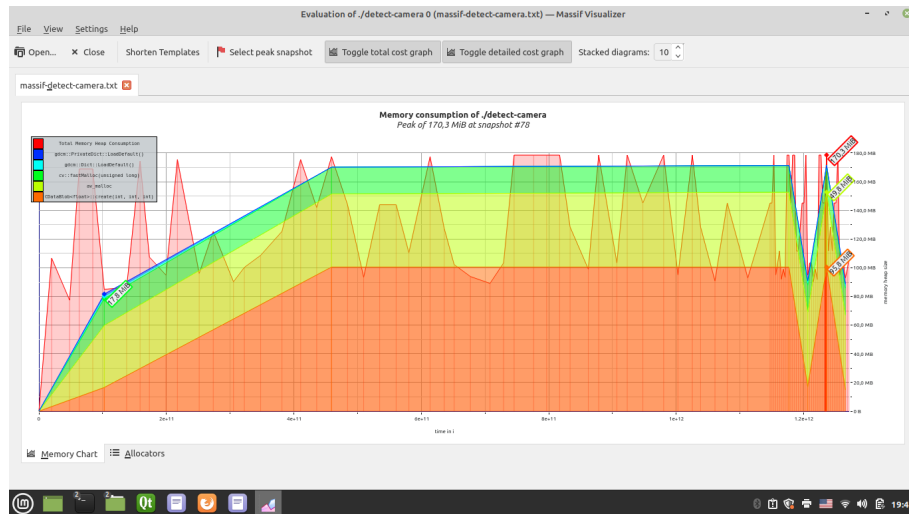


Слика 8: Коришћење меморије у програму *detect – image*

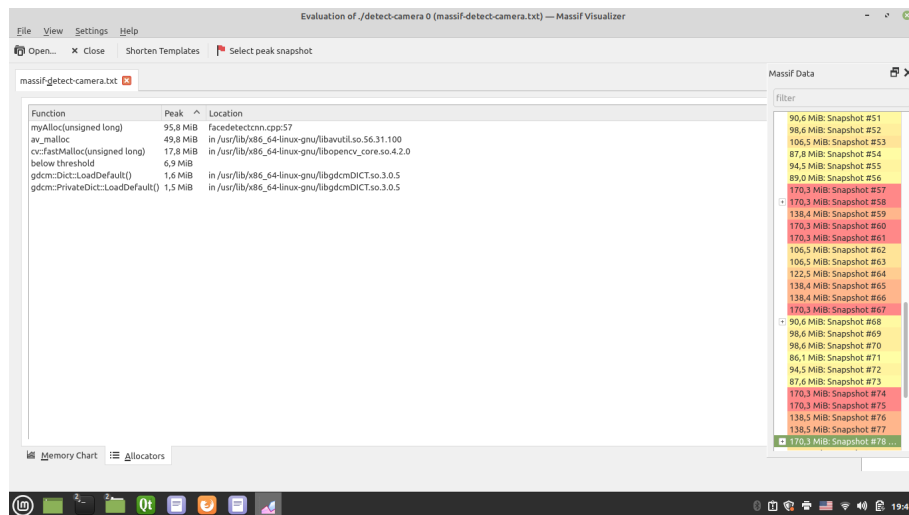


Слика 9: Алокатори у програму *detect – image*

На слици 8 можемо видети стање *heap* – а током извршавања програма *detect – image*. Видимо да се на почетку извршавања врши алоцирање меморије која ће се користити у наставку, и *peak* се дешава у неколико првих *snapshotova*. Након тога се само вршу деалокација меморије. Један од графика са слике који има велике скокове представља укупну коришћену меморију (заједно са стеком), па има доста промена.



Слика 10: Коришћење меморије у програму *detect – camera*



Слика 11: Алокатори у програму *detect – camera*

На слици 10 је приказао стање *heapa* током извршавања програма *detect – camera*. Ситуација је слична као код програма *detect – image*. Након детекције лица на једној слици, врши се деалокације. Тај случај је ухваћен у оквиру *snapshot#68*.

На основу горенаведених коментара можемо закључити да је одговорно коришћење *heap*.

1.3 Статичка анализа

1.3.1 *CppCheck*

CppCheck је алат за статичку анализу *C/C++* кода, који пружа јединствену анализу за детекцију грешака, са фокусом на недефинисано понашање и са опасним конструктима кода. Анализа добијена *CppCheck* није ни сагласна, нити је потпуна. Дакле, може имати и лажно позитивне резултате, као и лажно негативне. Могуће поруке:

- Грешка - недефинисано понашање (цурење меморије или цурење ресурса)
- Стил - редундантност, некоришћене функције/променљиве, потенцијалне грешке
- Перформансе - поправка ових порука не гарантује убрзање (јер је статичка анализа у питању)
- Преносивост
- Конфигурационе информације

Инсталирање и покретање алата: Детаљна инсталација званичној страници. Детаљан опис начина коришћења алата се може наћи у [2].

Приликом инсталације на Дебиан дистрибуцији:

```
sudo apt-get install cppcheck
```

Покретање алата *cppcheck* над пројектом *libfacedetection*:

```
cppcheck --enable=all --output-file="cppCheckOut.xml" --xml  
--inconclusive libfacedetection/
```

Додатни флагови:

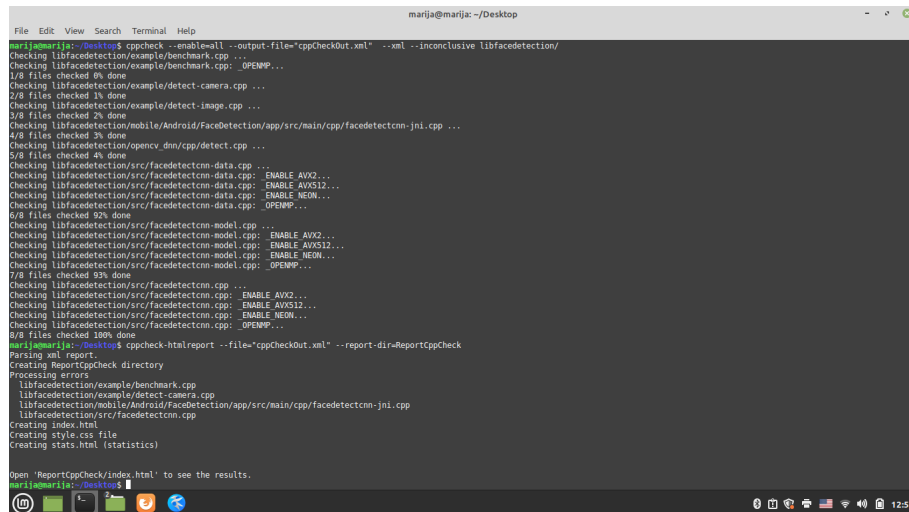
- *enable = all* - проналазак сви грешака
- *output - file* - дефинисање фајла у који се уписује
- *xml* - поруке у *xml* формату
- *inconclusive* - неуверљиве грешке (потенцијални *false positive*)

У оквиру алата је могуће направити *HTMLreport* од излазне поруке сачуване у *xml* формату.

```
cppcheck-htmlreport --report-dir=CppCheckReport --output-file="cppCheckOut.xml"
```

Скрипта за покретање алата се налази на *github*; у оквиру *README*.

Приказ покретања у алата: 12



```
marja@marja: ~/Desktop
File Edit View Search Terminal Help
marja@marja:~/Desktop$ cppcheck --enable=all --output-file="cppCheckOut.xml" --xml --inconclusive libfacedetection/
Checking libfacedetection/example/benchmark.cpp ...
Checking libfacedetection/example/benchmark.cpp: OPENMP...
1/8 files checked 0% done
Checking libfacedetection/example/detect-camera.cpp ...
Checking libfacedetection/example/detect-camera.cpp: ...
2/8 files checked 1% done
Checking libfacedetection/example/detect-image.cpp ...
Checking libfacedetection/example/detect-image.cpp: ...
3/8 files checked 2% done
Checking libfacedetection/mobile/Android/FaceDetection/app/src/main/cpp/facedetectcnn-jni.cpp ...
Checking libfacedetection/mobile/Android/FaceDetection/app/src/main/cpp/facedetectcnn-jni.cpp: ...
4/8 files checked 3% done
Checking libfacedetection/opencv_dnn/cpp/detect.cpp ...
Checking libfacedetection/opencv_dnn/cpp/detect.cpp: ...
5/8 files checked 4% done
Checking libfacedetection/src/facedetectcnn-data.cpp ...
Checking libfacedetection/src/facedetectcnn-data.cpp: ENABLE AVX2...
6/8 files checked 5% done
Checking libfacedetection/src/facedetectcnn-data.cpp: ENABLE AVX512...
Checking libfacedetection/src/facedetectcnn-data.cpp: ENABLE NEON...
7/8 files checked 6% done
Checking libfacedetection/src/facedetectcnn-data.cpp: OPENMP...
8/8 files checked 7% done
Checking libfacedetection/src/facedetectcnn-model.cpp ...
Checking libfacedetection/src/facedetectcnn-model.cpp: ENABLE AVX2...
Checking libfacedetection/src/facedetectcnn-model.cpp: ENABLE AVX512...
Checking libfacedetection/src/facedetectcnn-model.cpp: ENABLE NEON...
9/8 files checked 8% done
Checking libfacedetection/src/facedetectcnn-model.cpp: OPENMP...
10/8 files checked 9% done
Checking libfacedetection/src/facedetectcnn.cpp ...
Checking libfacedetection/src/facedetectcnn.cpp: ENABLE AVX2...
Checking libfacedetection/src/facedetectcnn.cpp: ENABLE AVX512...
Checking libfacedetection/src/facedetectcnn.cpp: ENABLE NEON...
11/8 files checked 10% done
Checking libfacedetection/src/facedetectcnn.cpp: OPENMP...
12/8 files checked 100% done
marja@marja:~/Desktop$ cppcheck-htmreport --file="cppCheckOut.xml" --report-dir=ReportCppCheck
Parsing xml report.
Creating ReportCppCheck directory
Processing errors
libfacedetection/example/benchmark.cpp
libfacedetection/example/detect-camera.cpp
libfacedetection/mobile/Android/FaceDetection/app/src/main/cpp/facedetectcnn-jni.cpp
libfacedetection/src/facedetectcnn-data.cpp
libfacedetection/src/facedetectcnn-model.cpp
libfacedetection/src/facedetectcnn.cpp
Creating index.html
Creating style.css file
Creating stats.html (statistics)

Open 'ReportCppCheck/index.html' to see the results.
```

Слика 12: Порука о грешци у *cppCheck*

Резултати анализе: На слици 13 се налази статистика анализе.

Cppcheck report - [project name]: Statistics

Back to summary	Top 10 files for error severity, total findings: 1
	1 libfacedetection/example/detect-camera.cpp
	Top 10 files for style severity, total findings: 5
	3 libfacedetection/example/benchmark.cpp
	1 libfacedetection/src/facedetectcnn.cpp
	1 libfacedetection/mobile/Android/FaceDetection/app/src/main/cpp/facedetectcnn-jni.cpp

Cppcheck 1.90 - a tool for static C/C++ code analysis

Internet: <http://cppcheck.net>
IRC: <irc://irc.freenode.net/cppcheck>

Слика 13: Статистика анализе алата *cppcheck*

На основу статистике видимо да је пронађена једна порука типа *грешка*, и 5 порука типа *стил*.

Детаљније о порукама се може видети на слици 14.

Cppcheck report - [project name]:				
Defect summary		Line	Id	Message
<input checked="" type="checkbox"/> Toggle all			missingInclude	Cppcheck cannot find all the include files (use --check-config for details)
Show # Defect ID				
<input checked="" type="checkbox"/> 2 unreadVariable		45	constArgument	Argument '131072*num_thread' to function malloc is always 131072
<input checked="" type="checkbox"/> 1 constArgument		57	unreadVariable	Variable 'pResults' is assigned a value that is never used.
<input checked="" type="checkbox"/> 1 constParameter		71	unreadVariable	Variable 'pResults' is assigned a value that is never used.
<input checked="" type="checkbox"/> 1 funcArgNamesDifferent				
<input checked="" type="checkbox"/> 1 memleak		76	memleak	Memory leak: pBuffer
<input checked="" type="checkbox"/> 1 missingInclude		15	unusedFunction	The function 'Java_org_dp_facedetection_MainActivity_facedetect' is never used.
<input checked="" type="checkbox"/> 1 unusedFunction		353	constParameter	Parameter 'inputoutputData' can be declared with const
8 total		661	funcArgNamesDifferent	Function 'kps_decode' argument 1 names different: declaration 'bbox_pred' definition 'kps_pred'

[Cppcheck](#) 1.90 - a tool for static C/C++ code analysis
[Internet: http://cppcheck.net](http://cppcheck.net)
[IRC: irc://irc.freenode.net/cppcheck](irc://irc.freenode.net/cppcheck)

Слика 14: Извештај алата *cppcheck*

Грешка се налази у 76 линији у фајлу *detect – camera.cpp*. Део кода који изазива грешку је дат у наставку:

```

55
56     int * pResults = NULL;
57     //pBuffer is used in the detection functions.
58     //If you call functions in multiple threads, please create one buffer for each thread!
59     unsigned char * pBuffer = (unsigned char *)malloc(DETECT_BUFFER_SIZE);
60     if(!pBuffer)
61     {
62         fprintf(stderr, "Can not alloc buffer.\n");
63         return -1;
64     }
65
66
67     VideoCapture cap;
68     Mat im;
69
70     if( isdigit(argv[1][0]))
71     {
72         cap.open(argv[1][0]-'0');
73         if(! cap.isOpened())
74         {
75             cerr << "Cannot open the camera." << endl;
76             return 0;
77         }

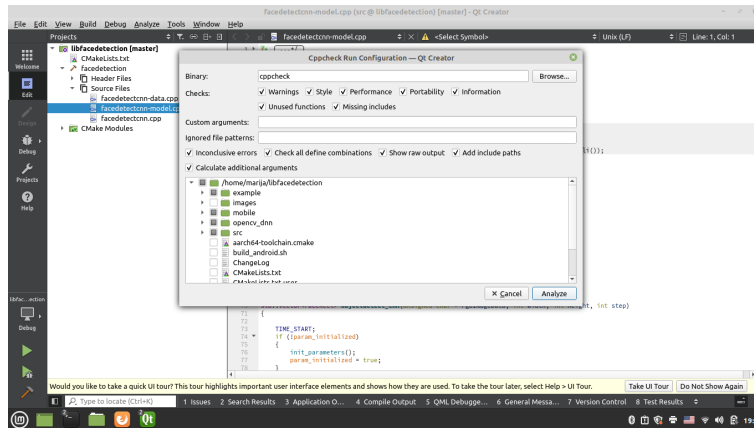
```

Дакле, меморија које је алоцирана у линији 59 се не ослобађа пре прекида извршавања програма, уколико је *VideoCapture* неуспешно отворен.

Поред тога су пријављена упозорења за некоришћене променљиве и функције. Детаљан извештај се може пронаћи у оквиру репозиторијума за анализу пројекта, у оквиру фолдера *CppCheckReport*. У наведеном репозиторијуму је за сваки фајл у коме је алат пронашао неки проблем направљен *html* извештај са обележеним местима са пронађеним проблемима.

Такође, могуће је покретање алата у оквиру развојног окрижења *QtCreator*. Пошто није **Help->About Plugins->Code Analyzer**. Изабрати *Cppcheck*. Након тога је неопходно рестартовање окружење.

Након тога: **Analyze -> Cppcheck** након чега се отвара прозор приказан на слици 15.



Слика 15: Покретање алата у оквиру *QtCreator*

Селектовањем поља *inconclusive errors* се пријављују и лажна упозорења (*false positive*).

2 Закључак

У оквиру пројекта је анализирана библиотека за детекцију лица и два њена примера употребе - детекција лица на прослеђеној слици и детекција лица са камере уређаја. Коришћена су четири алата за анализу. На основу њих можемо закључити да библиотека нема већих пропуста. Динамичком анализом нису пронађени пропуси, док је статичка анализа пронашла цурење меморије и неколико некоришћених променљивих.

3 Покретање скрипти за репродуковање резултата

```
git clone https://github.com/MATF-Software-Verification/2023_Analysis_libfacedetection
cd 2023_Analysis_libfacedetection/libfacedetection
git submodule init
git submodule update
```

Превођење библиотеке

```
cd libfacedetection
mkdir build
cd build
cmake .. -DCMAKE_INSTALL_PREFIX=install
-DDEBUG_SHARED_LIBS=ON -DCMAKE_BUILD_TYPE=Debug -DDEMO=OFF
cmake --build . --config Debug
cmake --build . --config Debug --target install
```

3.1 Покретање алата

Пре покретања алата потребно је преузимање пројекта и превођење библиотеке.

Након тога је потребно превести програме *detect – image* и *detect – camera*:

```
cd ../TestSamples
bash build.sh
```

CppCheck

```
cd ../CppCheckReport
bash run_cppcheck.sh
```

Gcov

```
cd ../CppCheckReport
bash run_gcov.sh
```

Memcheck

```
cd ../Memcheck
bash run_memcheck.sh
```

Massif

```
cd ../Massif
bash run_massif.sh
```

Литература

- [1] Ana Vulović Ivan Ristović. Verifikacija softvera skripta sa vežbi.
- [2] Cppcheck team. Cppcheck manual.