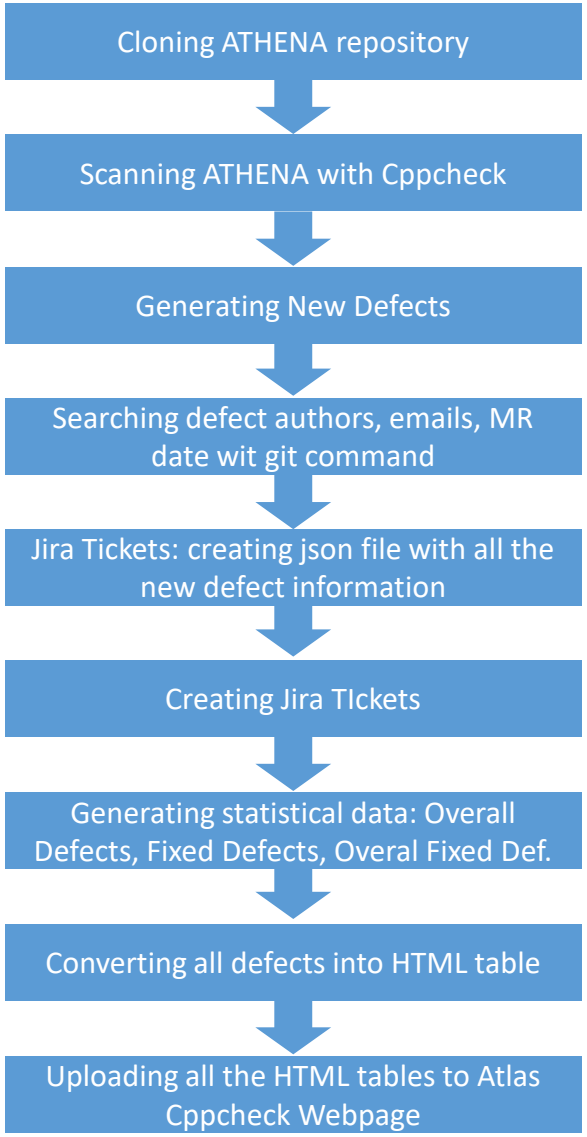
CPPcheck and Coverity Scanners

SHARMAZANASHVILI Alexander
Georgian Technical University

- Cppcheck Scan

- Cppcheck is lightweight open source application which can analyze C++ files considerably faster than any other static analysis tools
- This Scanning process consists of 9 consecutive steps
- Based on this scanning process automation tool was created
- Automation is written on bash, C++, JavaScript languages.
- all the steps in Automation are done without user interaction



Automation of Full ATHENA scan with Cppcheck

- To execute the automation tool we need to run one bash file auto.sh. This file runs automation steps such as generating defects xml file. Converting xml data to html table, creating Jira tickets for each defect.

```
#!/bin/bash


source config.sh

#Scanning timestamp
printf "\n[$TIMESTAMP] -- scan date" | tee -a $log_scandate

#pulls athena, starts cppcheck scanning, compares defect files, gets defects info
g++ -std=c++11 automate.cpp -o automate.out
./automate.out | tee $log_auto

#converts xml to html
./convertor.sh | tee -a $log_auto

#checks if defects exist. if yes creates jira tickets
g++ -std=c++11 check_defects.cpp -o check_defects.out
./check_defects.out | tee -a $log_auto
```

- For defect generating we have automate.cpp file
- For XML to HTML data conversion we have convertor.sh file
- For Jira ticket creation we have check_defects.cpp file.

- With automate.cpp file, we clone or pull ATHENA repository from Gitlab. Scan it with Cppcheck, generate new defects with defect filter application which is based on Nodejs and run git log to get author name, email, MR date for each detected defect.

automate.cpp

```
                string name = exec("git log -1 --pretty=format:%an --  "+ str);
                string mail = exec("git log -1 --pretty=format:%ae --  "+ str);

                chdir("..");

                if(line.find("<error ") != string::npos ){

                    line.insert( line.length()-2," scan_date=\""+scan_date+"\" ");
                    line.insert( line.length()-2," mrdate=\"" +mrdate+ "\" ");
                    line.insert( line.length()-2," author=\"" +name+ "\" ");
                    line.insert( line.length()-2," mail=\"" +mail+ "\" ");

                    datastring += line +"\n";
                }

            }
        }

        data = datastring;
        write(writefile, data);
}

int main()
{

    Gitclone gitclone;

    cout<<"\n ======= Phase 3: ======= "<<endl;
    cout<<" ======= Created results.xml file ======= "<<endl;

        //vadarebt mimdinare da wina kviris skanirebis failebs
        exec("scl enable rh-nodejs12 'node main.js'");
    //  exec("node main.js");
        exec("scl enable rh-nodejs12 'node compare.js'");
    //  exec("node compare.js");

    cout<<" ======= Done ======= "<<endl;
```

Automation of Full ATHENA scan with Cppcheck

- New Defect detection application based on NodeJS contains 2 files. main.js and compare.js
- With main.js we read cppcheck generated defect file, get defect Path, Line, defect message and modify it into MultiMap Data Structure. Output from main.js file is two JSON file. One newDefects JSON file and one oldDefetcs JSON file

- With compare.js we read JSON files generated by main.js and do Map to Map defect comparison. Where Map key value is File path. If two defect file paths are MATCHED compare.js starts comparing Map values. Map values are defect message and defect line. If any on the values are different compare.js identifies it as new defect.

compare.js

```
GetNewDefects() {
    var storage = [];
    for( var i in this.new) {

        var dnew = this.new[ i ];
        var dold = this.old[ i ]; //get new defect file in old defect file.

        if( dold == undefined ) {
            storage.push( dnew );
            continue;
        }
        // console.log( this.new[i] )
        for( var k = 0; k < dnew.length; k++) {
            var found = false;
            for( var z = 0; z < dold.length; z++) {
                /*
                    check 3: if message is different
                */
                if( dnew[k].message == dold[z].message) {
                    /*
                        enable for line difference
                    */
                    // if(dnew[k].line == dold[z].line) {found = true; break; }
                    found = true;
                    break;
                }
            }
            if(!found) {

                // if( storage.indexOf(dnew[k]) > -1 ) { continue; }
                storage.push(dnew[k]);
            }
        }
    }
    /*
        return stored data
    */
    return storage;
}
```

Automation of Full ATHENA scan with Cppcheck

- before we create JIRA tickets we need to check if new defects were generated at all. For this we have to check new defects xml file results.xml.
- For this was created check_defects.cpp file. This file checks if results.xml exists, contains any data at all else it'll run Jiraticketcreator.sh bash file.

```cpp
int main(){

    std::ifstream file("results.xml");

    if(!file){

        cout << "results.xml not found"<<endl;
        // file is not open
        return 1;
    }

    if ( file.peek() == std::ifstream::traits_type::eof()){

        cout<<"We don't have any Defects today"<<endl;

        return 2;
    }

//file is open and isn't empty
//cout<<"Continue automation"<<endl;

    string run_jira = "./jiraticketcreator.sh";
    std::system(run_jira.c_str());


    return 0;
}
```

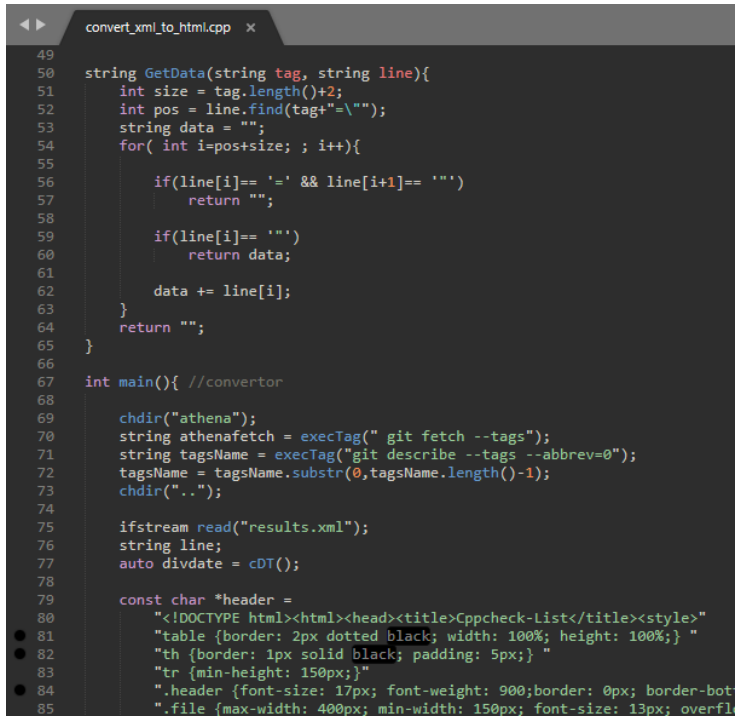
- To create JIRA tickets, first we need search for defect author username to be able to assign tickets automatically. This is done by ldap.sh bash file. This bash file contains LDAPsearch command for defect authors. Output is ldap.txt file. we read ldap.txt file extract author usernames and pass it as argument to jiradatamaker.cpp

- Jiradatamaker.cpp file creates txt file with json format which contains all the necessary data for JIRA ticket creation.
- All is left to run cern-get-sso-cookie for authentication And post generated JIRA data file with CURL

```
#!/bin/bash

chmod +x jiraticketcreator.sh

#creates ldap.sh file for username search
g++ -std=c++11 ldapshellcreator.cpp -o ldapshellcreator.out
./ldapshellcreator.out


chmod a+x ldap.sh
#runs username search script
./ldap.sh

#modifies and adds username to diff.xml
g++ -std=c++11 LdapReader.cpp -o LdapReader.out
./LdapReader.out

#creates data for curl request
g++ -std=c++11 jiradatamaker.cpp -o jiradatamaker.out
./jiradatamaker.out

g++ -std=c++11 eraselastchar.cpp -o eraselastchar.out
./eraselastchar.out

#cookie based authentication for jira website
cern-get-sso-cookie -u https://its.cern.ch/jira/loginCern.jsp -o jira.txt
#curl post request
curl -b jira.txt \
    -X POST  https://its.cern.ch/jira/rest/api/2/issue/bulk \
    --data @jj.txt \
    -H "Content-Type:application/json" \
    -i
```

Automation of Full ATHENA scan with Cppcheck

- For XML to HTML data conversion we have one bash file convertor.sh. This bash file compiles and runs convert_xml_to_html.cpp file. This file converts All the generated defect files in automation tool: new defects, overall defects, fixed defects and overall fixed defects.



- After the first 8 step of the automation are done, in order to updoad all the HTML tables into Atlas cppcheck webpage we need commit and push all the changes into automation tool Gitlab repository.
- Pushing all the changes into master branch triggers webhook event, which is uploading everything into atlas cppcheck webpage
- Gitlab repository: https://gitlab.cern.ch/atlas-sit/cpp_check

Results of Full ATHENA scan with Cppcheck

1. JIRA TICKETS

Full Defect List Site	
ID	10022021201
Scan Date	10-02-2021
Mr Date	11-11-2020
File	athena/Reconstruction/Jet/JetUncertainties/util/M
Line	201
Defect Message	Invalid log() argument nr 1. The value is 0 but the valic
Author	Tadej Novak
Mail	tadej.novak@cern.ch

Attachments

Activity

All Comments Work Log

William Keaton Balunas

Fixed in !42505

Edit

Full Defect List Site	
ID	10022021181
Scan Date	10-02-2021
Mr Date	06-01-2021
File	athena/PhysicsAnalysis/TopPhys/xAOD/TopPartons/Root/CalcTTZPartonHistory.cxx
Line	174
Defect Message	Uninitialized variable: partonAuxCont
Author	Nils Krumnack
Mail	krumnack@iastate.edu

Attachments

Drop files to attach, or browse.

Activity

All Comments Work Log History Activity

Tomas Dado added a comment - 10/Feb/21 4:58 PM

Hi, I am confiused about the error, the line that the check points to is:

`xAOD::PartonHistoryAuxContainer* partonAuxCont = new xAOD::PartonHistoryAuxContainer {};` shouldnt that call a default constructor?

Cheers,
Tomas

Full Defect List Site	
ID	
Scan Date	
Mr Date	
File	athena/Recon
Line	
Defect Message	Out of bounds access
Author	
Mail	

Attachments

Drop files to attach, or browse.

Activity

All Comments Work Log History Activity

William Keaton Balunas added a comment - 12/Apr/21 1:05 PM

JetByVertexFinder was recently made obsolete and removed in !42301, so I think it's safe to assume this is no longer a problem.

Edit

Full Defect List Site	Cppcheck version 2.3
ID	250320211
Scan Date	25-03-2021
Mr Date	24-03-2021
File	athena/Trigger/TrigT1/TrigT1CaloFexPerf/src/EFexEMClusterTool.cxx
Line	751
Defect Message	Shifting by a negative value is undefined behaviour
Author	Ben Carlson
Mail	ben.carlson@cern.ch
Xml-Snipet	<error id="shiftNegative" severity="error" msg="Shifting by a negative value is undefined behaviour" verbose="Shifting by a negative value is undefined behaviour" cwe="758" hash="12721935486367181253"> <location file="athena/Trigger/TrigT1/TrigT1CaloFexPerf/src/EFexEMClusterTool.cxx" line="751" column="130"/> </error>

Attachments

Drop files to attach, or browse.

Issue Links

is related to

ATR-22750 Migration of L1Calo floating point simulation to rel22 OPEN

Activity

All Comments Work Log History Activity

Ben Carlson added a comment - 25/Mar/21 3:24 PM

Joerg Stelzer Jon Burr

Edit

Jon Burr added a comment - 25/Mar/21 3:33 PM

To me this must be a bug in a CppCheck. This is being triggered by the `operator<<` in a message statement. Obviously giving this a negative number is completely fine, but I would guess that CppCheck thinks that this is a bitshift operator, in which case it would be correct.

Edit

Joerg Stelzer added a comment - 12/Apr/21 5:24 PM

I agree Jon Burr . It is a bit puzzling though, this must happen in thousands of places in ATLAS code though. Maybe it is because of the previous number is a product of two and CppCheck gets confused by the operator precedence.

Results of Full ATHENA scan with Cppcheck

2. HTML Tables
https://atlas-cppcheck.web.cern.ch/

1. New defects

Updated:10-06-2021

ID	SCAN DATE	FILE	MR DATE	LINE	DEFECT MESSAGE	AUTHOR	MAIL
100620211	10-06-2021	athena/HighGranularityTimingDetector/HGTD_DetDescr/HGTD_GeoModel/src/HGTD_DetectorFactory.cxx	03-06-2021	1062	Out of bounds access in expression 'modulePositions.back()' because 'modulePositions' is empty and 'back' may be non-zero.	David Richard Shope	david.richard.shope@cern.ch

2. Overall defects

Updated:10-06-2021

ID	SCAN DATE	FILE	MR DATE	LINE	DEFECT MESSAGE	AUTHOR	MAIL
1	27-05-2020	athena/AtlasTest/ControlTest/test/StoreGateSvcClient_test.cxx	02-03-2020	258	Memory leak: x	Frank Winklmeier	frank.winklmeier@cern.ch
2	27-05-2020	athena/AtlasTest/DatabaseTest/AthenaDBTestRec/src/lib/TestCoolRecFolder.cxx	05-04-2016	222	syntax error	Peter Van Gemmeren	peter.van.gemmeren@cern.ch
3	27-05-2020	athena/AtlasTest/GoogleTestTools/test/gt_GoogleTestTools.cxx	18-12-2018	26	syntax error	Edward Moyse	edward.moyse@cern.ch
4	27-05-2020	athena/Calorimeter/CaloLocalHadCalib/src/GetLCWeights.cxx	11-02-2020	222	Syntax Error: AST broken, 'if' doesn't have two operands.	christos	christos@cern.ch
5	27-05-2020	athena/Calorimeter/CaloMonitoring/rootMacros/CellClusterLinkTool.C	11-04-2014	159	Uninitialized struct member: onebin.m_nx	Walter Lampl	Walter.Lampl@cern.ch
6	27-05-2020	athena/Calorimeter/CaloMonitoring/rootMacros/CellClusterLinkTool.C	11-04-2014	159	Uninitialized struct member: onebin.m_ny	Walter Lampl	Walter.Lampl@cern.ch

3. Fixed defects

Updated:10-06-2021

ID	SCAN DATE	FILE	MR DATE	LINE	DEFECT MESSAGE	AUTHOR	MAIL
1	10-06-2021	athena/InnerDetector/InDetCalibAlgs/TRT_CalibAlgs/share/CalibrateTRT.cpp	07-06-2021	898	Uninitialized struct member: startdata.det	Shaun Roe	shaun.roe@cern.ch
2	10-06-2021	athena/InnerDetector/InDetCalibAlgs/TRT_CalibAlgs/share/CalibrateTRT.cpp	07-06-2021	898	Uninitialized struct member: startdata.lay	Shaun Roe	shaun.roe@cern.ch
3	10-06-2021	athena/InnerDetector/InDetCalibAlgs/TRT_CalibAlgs/share/CalibrateTRT.cpp	07-06-2021	898	Uninitialized struct member: startdata.mod	Shaun Roe	shaun.roe@cern.ch
4	10-06-2021	athena/InnerDetector/InDetCalibAlgs/TRT_CalibAlgs/share/CalibrateTRT.cpp	07-06-2021	898	Uninitialized struct member: startdata.brd	Shaun Roe	shaun.roe@cern.ch
5	10-06-2021	athena/InnerDetector/InDetCalibAlgs/TRT_CalibAlgs/share/CalibrateTRT.cpp	07-06-2021	898	Uninitialized struct member: startdata.chp	Shaun Roe	shaun.roe@cern.ch
6	10-06-2021	athena/InnerDetector/InDetCalibAlgs/TRT_CalibAlgs/share/CalibrateTRT.cpp	07-06-2021	898	Uninitialized struct member: startdata.stl	Shaun Roe	shaun.roe@cern.ch

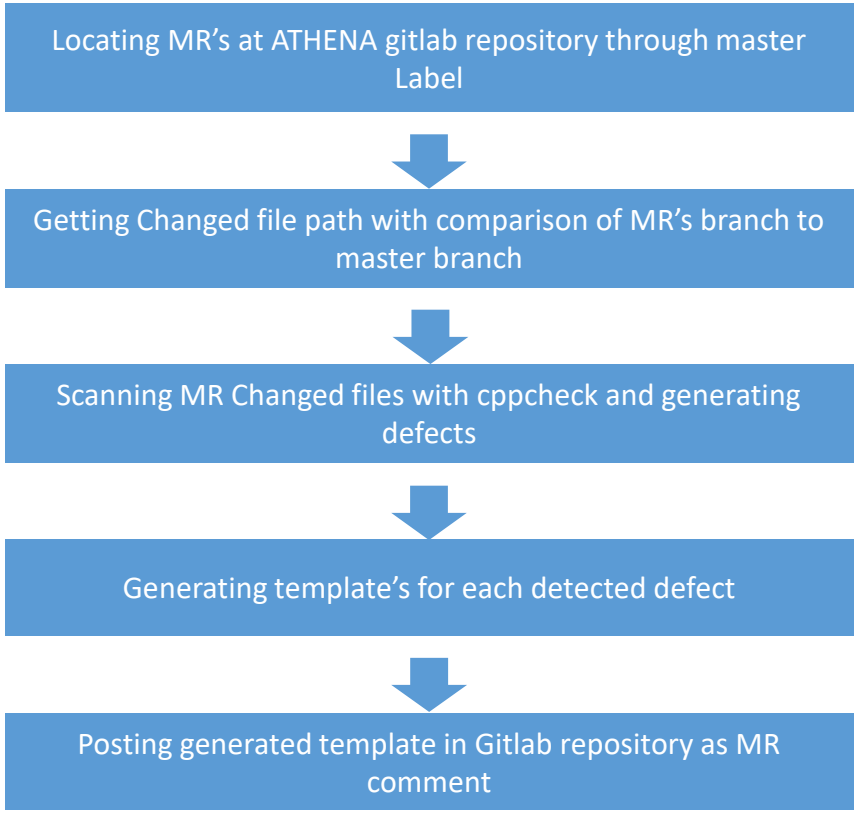
4. Overall Fixed defects

Updated:10-06-2021

ID	SCAN DATE	FILE	MR DATE	LINE	DEFECT MESSAGE	AUTHOR	MAIL
1	27-05-2020	athena/MuonSpectrometer/MuonConditions/MuonCondGeneral/MuonCondCool/src/CscReadWriteCoolStr.icc	27-05-2020	256	Shifting by a negative value is undefined behaviour	Nicolas Koehler	nicolas.koehler@cern.ch
2	09-06-2020	athena/InnerDetector/InDetRawAlgs/InDetOverlay/test/BCMOverlay_test.cxx	09-06-2020	56	syntax error	Jakob Novak	jakob.novak@cern.ch
3	23-06-2020	athena/Trigger/TrigT1/TrigT1CaloMonitoring/src/CpmMonitorAlgorithm.cxx	19-06-2020	205	Uninitialized struct member: monTobRoI.isolationBitSet	Paul Daniel Thompson	paul.daniel.thompson@cern.ch
4	23-06-2020	athena/Trigger/TrigT1/TrigT1CaloMonitoring/src/CpmMonitorAlgorithm.cxx	19-06-2020	311	Uninitialized struct member: monCmxCpTob.ybase	Paul Daniel Thompson	paul.daniel.thompson@cern.ch

Individual Merge Request scan with Cppcheck

- This is scanning process where we run Cppcheck on Merge Request's from ATHENA Gitlab repository
- This scanning process consists of 5 consecutive steps
- Individual MR scanning process is fully automated
- It is written on python and bash
- This is automation tool is executed through one bash file.



Automation of Individual Merge Request scan

- To start individual MR scanning process we need to make sure that we have all the necessary applications and library's are installed. for this we need to run ./start.sh setup. this command will check if the python3 and cppcheck are installed, if not bash file will install them.

```
function setup(){

        packages=("python3" "cppcheck")

        for pkg in ${packages[@]}; do

                is_pkg_installed=$(dpkg-query -W --showformat='${Status}\n' ${pkg} | grep "install ok installed")

                if [ "${is_pkg_installed}" == "install ok installed" ]; then
                        echo ${pkg} is installed.
                else
                        echo "INSTALLING"
                        apt-get install -y python3
                        apt-get install -y cppcheck
                fi

        done


        if [ 'pip list | grep -F python-gitlab' ];
        then
                echo "python-gitlab is installed"
        else
                echo "installing"
                pip3 install python-gitlab
        fi


        if [ 'pip list | grep -F xmltodict' ];
        then
                echo "xmltodict is installed"
        else
                echo "installing"
                pip3 install xmltodict
        fi

}
```

- after setup we can run individual MR scanning process with ./start.sh scan. this will launch python file autoMR.py to start scanning process.
- At terminal autoMR.py will generate all the MR's with master label their id, branch, title and ask us to type MR id
- We can choose any of the MR's iid listed in terminal and type it
- Automation will compare MR branch and master branch to find changed files with:

git diff --diff-filter=ACM --name-only master... -- *.cpp *.cxx *.h'

- Output of this command is passed to cppcheck to scan and generate defects
- After defects are generated each of them are modified and written into template's
- At last this template's are appended as MR's comment

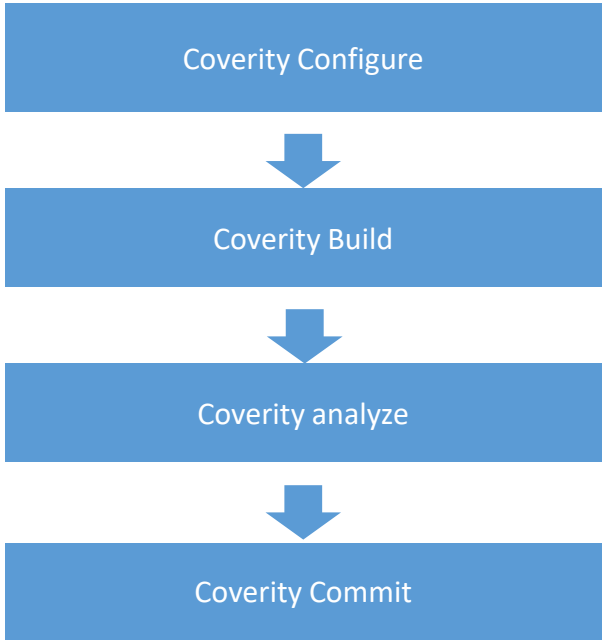
This is example of the template

Cppcheck Defect Report	
file	src/b.cpp
line	4
msg	"Unmatched '{'. Configuration: ''."

- Coverity Scan

Coverity Scan

- Coverity is much more complex static analysis application than Cppcheck. In comparison Coverity is generating much detailed defects then cppcheck
- The main advantage of Coverity is his platform Coverity Connect. This is Coverity database where all the defect information are appended
- All defects have unique identifier which were saved in Coverity database. These identifiers give us possibility to look after specific defect even if it is located on different line
- In order to generate Defects from Coverity, we need to do 4 consecutive steps



Coverity Scan

- Coverity Configure is the first step of Coverity scan. With this step we are configuring compiler for the Coverity build. In order to build C++ files at ATHENA repository we need to configure GCC compiler.
- Coverity build is the second step of Coverity scan
  - In the past ATHENA build was our biggest Challenge. But THANKS to Atilla we successfully generated Build Configuration file for Athena Build.
  - After Build Configuration file is generated, all is left to run Coverity Build command

```
asetup AthenaExternals,master,latest

mkdir athena-build
cd athena-build/
cmake ../athena/Projects/Athena/ 2>&1 | tee /build/luka/cmake_config.log

$COVBUILD --dir $COVDIR make 2>&1 | tee /build/luka/cmake_build.log
```

$COVBUILD – this is a Coverity command cov-build

--dir $COVDIR – this is a directory where ATHENA build will be saved

Make – this is build command

- Coverity analyze is the Third step of Coverity scan. Here we run Coverity analyze command on build directory for checking all the possible defects in the source code

Coverity Scan

- Coverity analyze command is:

  $COVANALYZE --dir $COVBUILDDIR --strip-path $STRIPPATH --all

  - $COVANALYZE – is the Coverity analyze command cov-analyze
  - --dir $COVBUILDDIR - is the Coverity Build directory
  - --strip-path $STRIPPATH – is better usage of Coverity connect
  - --all – means that all the possible checkers are enabled for Coverity analyze command
  - For example: --concurrency, --security, --enable-parse-warnings, PARSE_ERROR

- Results from last Coverity scan 26/03/2022
  - Coverity Build took 8 hours and 50 minutes to finish
  - Coverity analyze took 3 hours and 30 minutes to finish
  - Files analyzed: 27 653
  - Functions analyzed: 359 175
  - Defects found: 12 495

Coverity Scan

- Last step in Coverity Scanning process is Coverity Commit step
- At this step Coverity reads analysis output and source data stored at analysis directory and writes them to Coverity Connect database by command:

```
$COVCOMMIT --url https://atlas-coverity.cern.ch --dir $COVBUILDDIR --stream $STREAM --user $USER --password $PASS
```

  - $COVCOMMIT - is Coverity command cov-commit-defects
  - --url https://atlas-coverity.cern.ch - with this Coverity command is connecting to the Coverity Connect database through URL
  - --dir $COVBUILDDIR - is directory where Coverity analyze output is located
  - --stream $STREAM - is just name of analyze output
  - --user $USER and --pass $PASS are just for specifying committer name at Coverity Connect database, password for authenticatiion

  - Coverity connect webpage is: https://atlas-coverity.cern.ch/

Thanks for your attention!