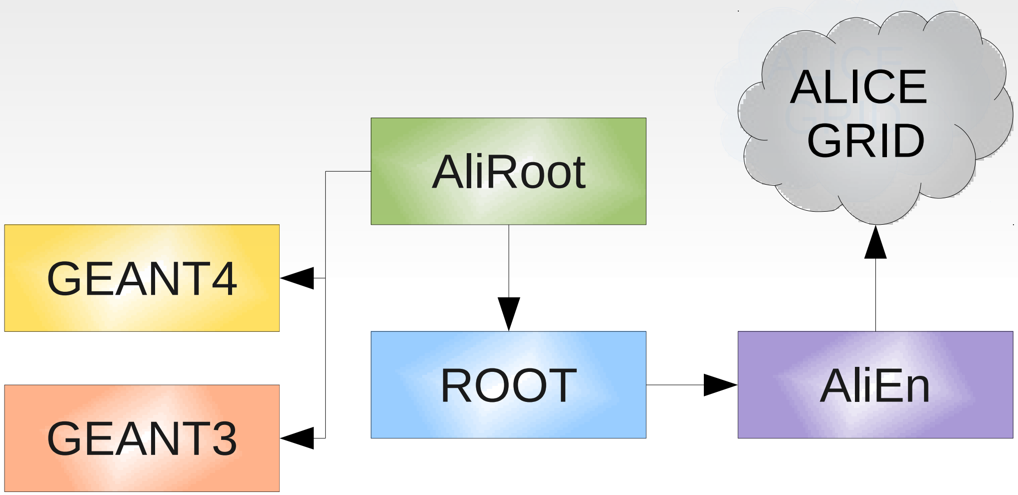
ALICE experience using Coverity

Federico Carminati, Peter Hristov, Axel Naumann and Olga Datskova (CERN)

presented by Olga Datskova (CERN, ALICE Offline)
Ferrara, 04/07/2011

AliRoot framework

AliRoot is the ALICE Off-line framework used for simulation, reconstruction, analysis and visualisation of experiment data. [1]



AliRoot is developed with the following dependencies in mind [2]:

- **ROOT** is the foundation framework upon which AliRoot is built.
- **AliEn** provides GRID support to users and sites.
- **GEANT3** and **GEANT4** are used for simulation and Monte-Carlo productions.

Main tasks for static C/C++ code analysis in AliRoot:

- Maintaining good quality of code with a large developer base.
- Ensuring rapid fixes in order to keep up with weekly release cycles.
- Debugging irreproducible problems.

Coverity analysis

> *“Coverity analysis solutions enable users to test their code against business policies and thresholds while in development. Finding and addressing defects early in the lifecycle saves developer’s time by minimizing rework and keeping releases on schedule.”* [3]

Coverity analysis suite consists of the following set of tools [3]:

- **Coverity Static Analysis** is a command line program for identifying program defects through static source code inspection. The tool supports C/C++, Java and C# programming languages.
- **Coverity Dynamic Analysis** performs dynamic evaluation of running code. The tool supports Java programming language.
- **Coverity Integrity Manager** is a web interface for managing defects, which were discovered by Coverity Static Analysis and Coverity Dynamic Analysis tools.

Coverity features

Coverity Static Analysis and **Integrity Manager** have the following features:

- The installation procedure for both distributions is very straight forward and is done through the provided installation scripts.
- **Coverity Static Analysis** tool provides the flexibility of a command line, allowing for greater control over the build and analysis processes.
- **Coverity Integrity Manager** has the following aspects:
  - Database for managing defects and users assigned to them. The database has the functionality to be queried securely for remote database administration.
  - Web server allows for a centralised web service, where users can log in and examine their code. The system supports LDAP authentication.

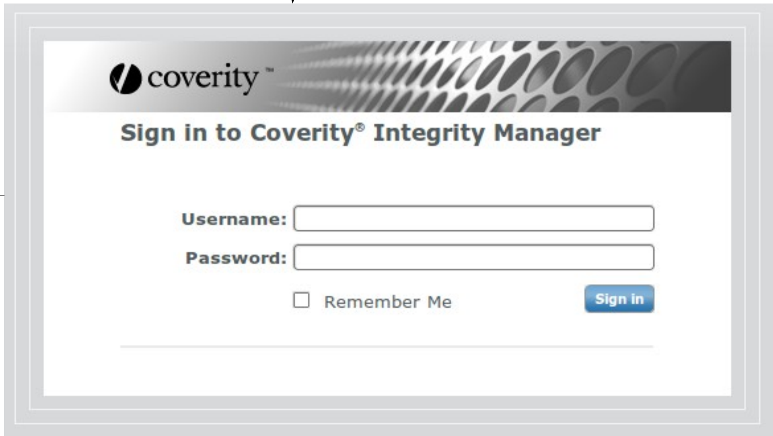
AliRoot static analysis setup

```
$ cd $ALICE_ROOT
$ svn update
$ cd $COVERITY_IN
$ cov-build --dir $COVERITY_OUT make
```

```
$ cov-analyze --dir $COVERITY_OUT [list of checkers] [options]
```

```
$ cov-commit-defects -dir $COVERITY_OUT [options]
```

users commit fixes



alicoverity log in screen (26/06/2011)

AliRoot has been set up to undergo consistent daily static analysis. The following steps describe the process:

1) AliRoot sources are updated with the latest development code and fixes.

2) The sources are then built through cov-build tool, producing intermediate code in $COVERITY_OUT

3) Analysis of the built sources can then commence. Here we specify the desired checks that the tool must perform.

4) After successful completion of analysis, the resulting reports are committed to the Coverity Integrity Manager.

5) The developer then checks through the web page for defects assigned to him/her and starts to work on the fixes as necessary.

Coverity maintenance and use policy

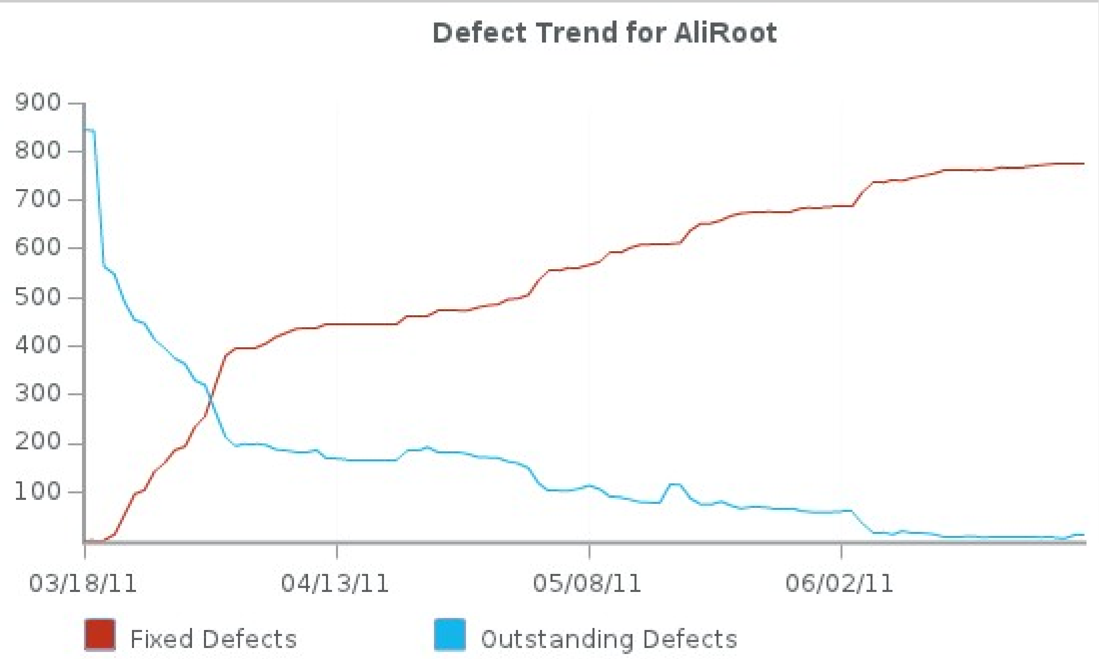
To achieve minimal maintenance and consistent performance from the Coverity server, the following steps were automated in cron:

- Coverity build, analyse and submit procedures.
- Retrieving and modifying defect information from the database, whether to send a notification or assign defect to the user.
- Performing daily backups of the Coverity database.

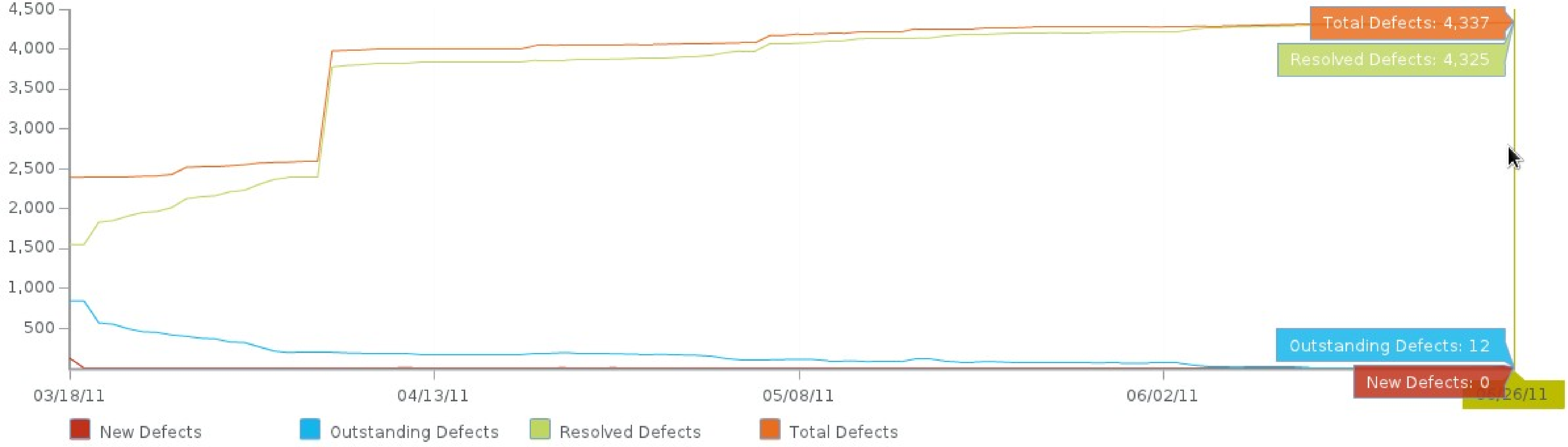
To ensure persistent quality of code, the following policies were introduced:

- **All** defects must be fixed promptly irrespective of their impact.
- “Top 10” users receive email notifications every day. The message also contains compilation warnings which should be taken care of quickly.
- Recent addition (comes into effect on the 11th of July): users have 7 days to fix all their defects, otherwise they are banned from subversion until they do.

Coverity results for AliRoot



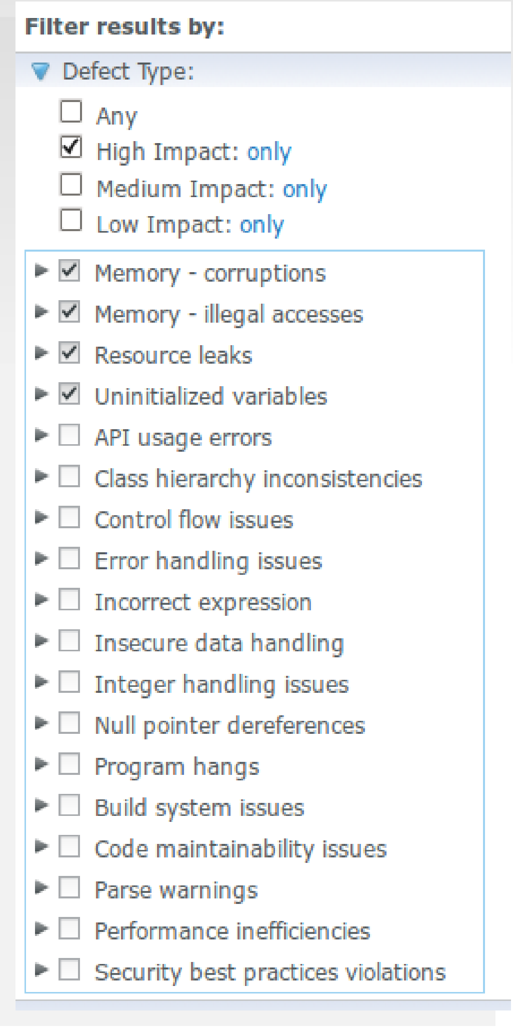
- Coverity was put into active use starting from January 2011.
- Approximately 6000 defects have been identified initially by Coverity.
- At present time, we have 12 defects with more fixes being committed and new code developed every day.



alicoverity Dashboard and metric report (26/06/2011)

Coverity defects

Coverity provides its own classification of discovered defects into appropriate impact categories as seen in the project side menu:

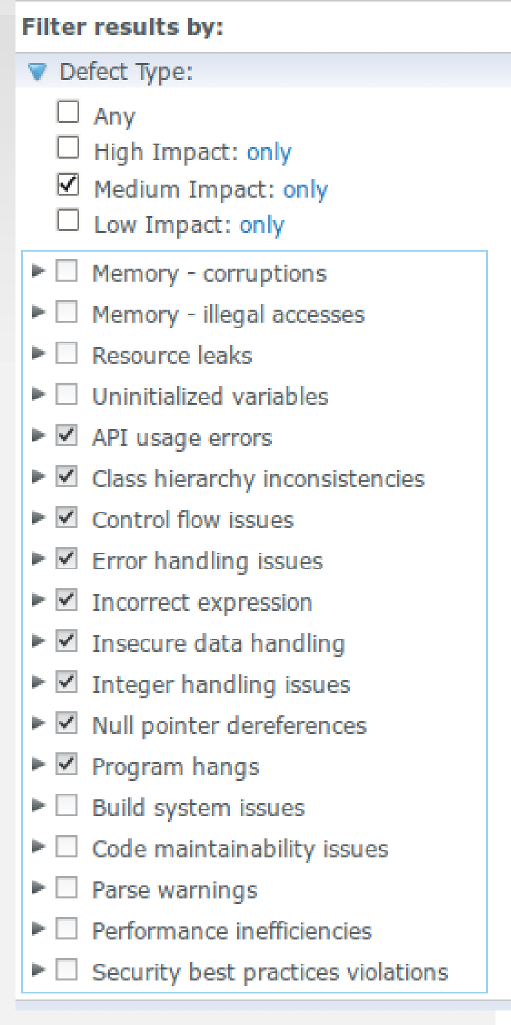


Filter results by:

Defect Type:

- □ Any
- ☑ High Impact: only
- □ Medium Impact: only
- □ Low Impact: only

- ☑ Memory - corruptions
- ☑ Memory - illegal accesses
- ☑ Resource leaks
- ☑ Uninitialized variables
- □ API usage errors
- □ Class hierarchy inconsistencies
- □ Control flow issues
- □ Error handling issues
- □ Incorrect expression
- □ Insecure data handling
- □ Integer handling issues
- □ Null pointer dereferences
- □ Program hangs
- □ Build system issues
- □ Code maintainability issues
- □ Parse warnings
- □ Performance inefficiencies
- □ Security best practices violations

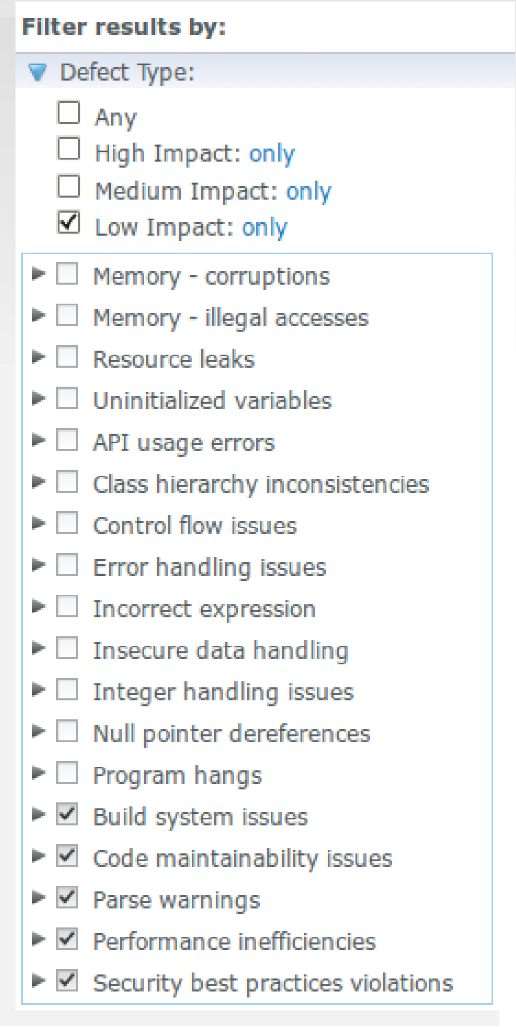


Filter results by:

Defect Type:

- □ Any
- □ High Impact: only
- ☑ Medium Impact: only
- □ Low Impact: only

- □ Memory - corruptions
- □ Memory - illegal accesses
- □ Resource leaks
- □ Uninitialized variables
- ☑ API usage errors
- ☑ Class hierarchy inconsistencies
- ☑ Control flow issues
- ☑ Error handling issues
- ☑ Incorrect expression
- ☑ Insecure data handling
- ☑ Integer handling issues
- ☑ Null pointer dereferences
- ☑ Program hangs
- □ Build system issues
- □ Code maintainability issues
- □ Parse warnings
- □ Performance inefficiencies
- □ Security best practices violations



Filter results by:

Defect Type:

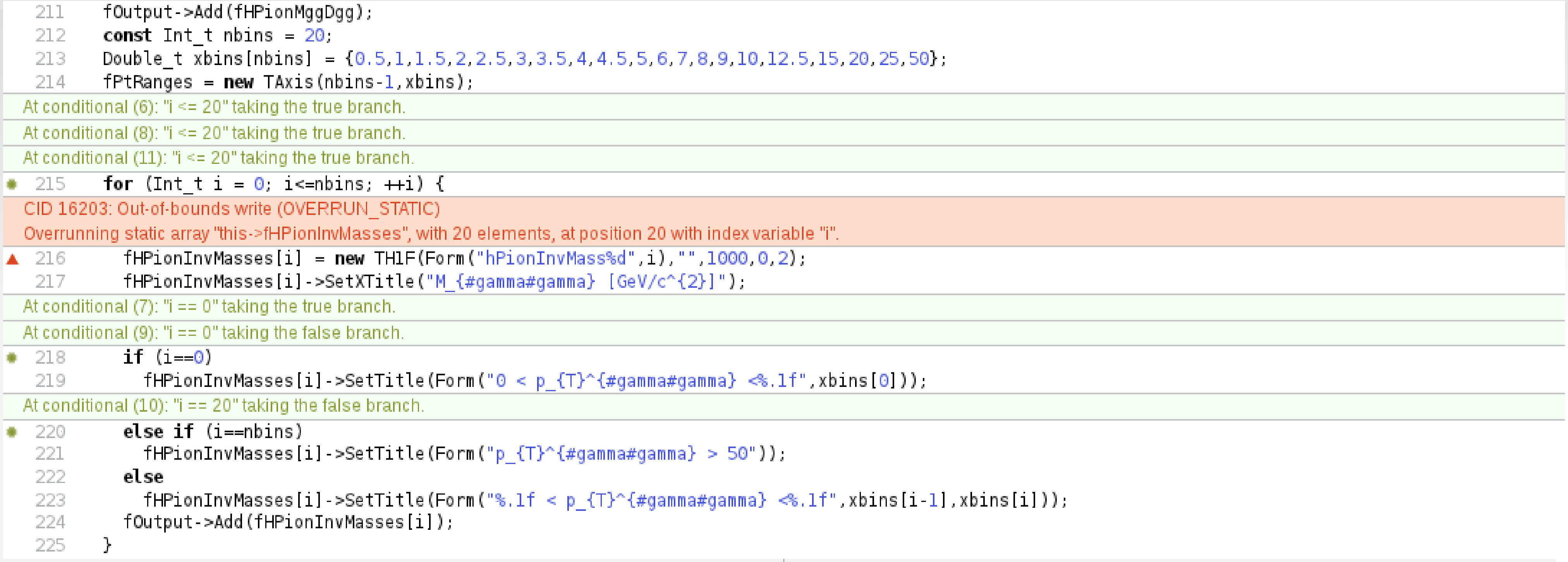
- □ Any
- □ High Impact: only
- □ Medium Impact: only
- ☑ Low Impact: only

- □ Memory - corruptions
- □ Memory - illegal accesses
- □ Resource leaks
- □ Uninitialized variables
- □ API usage errors
- □ Class hierarchy inconsistencies
- □ Control flow issues
- □ Error handling issues
- □ Incorrect expression
- □ Insecure data handling
- □ Integer handling issues
- □ Null pointer dereferences
- □ Program hangs
- ☑ Build system issues
- ☑ Code maintainability issues
- ☑ Parse warnings
- ☑ Performance inefficiencies
- ☑ Security best practices violations

alicoverity Coverity Integrity Manager menu (26/06/2011)

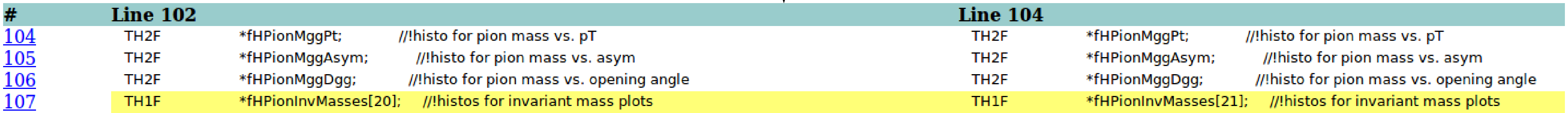
High impact defect: Out of bounds write [4]

Defect #16203: when i = 20, an attempt to write to 21st value of fHPionInvMasses will be made, which has been defined to have only 20 elements.



```
fOutput->Add(fHPionMggDgg);
const Int_t nbins = 20;
Double_t xbins[nbins] = {0.5,1,1.5,2,2.5,3,3.5,4,4.5,5,6,7,8,9,10,12.5,15,20,25,50};
fPtRanges = new TAxis(nbins-1,xbins);
At conditional (6): "i <= 20" taking the true branch.
At conditional (8): "i <= 20" taking the true branch.
At conditional (11): "i <= 20" taking the true branch.
for (Int_t i = 0; i<=nbins; ++i) {
CID 16203: Out-of-bounds write (OVERRUN_STATIC)
Overrunning static array "this->fHPionInvMasses", with 20 elements, at position 20 with index variable "i".
  fHPionInvMasses[i] = new TH1F(Form("hPionInvMass%d",i),"",1000,0,2);
  fHPionInvMasses[i]->SetXTitle("M_{#gamma#gamma} [GeV/c^{2}]");
At conditional (7): "i == 0" taking the true branch.
At conditional (9): "i == 0" taking the false branch.
  if (i==0)
    fHPionInvMasses[i]->SetTitle(Form("0 < p_{T}^{#gamma#gamma} <%.1f",xbins[0]));
At conditional (10): "i == 20" taking the false branch.
  else if (i==nbins)
    fHPionInvMasses[i]->SetTitle(Form("p_{T}^{#gamma#gamma} > 50"));
  else
    fHPionInvMasses[i]->SetTitle(Form("%.1f < p_{T}^{#gamma#gamma} <%.1f",xbins[i-1],xbins[i]));
  fOutput->Add(fHPionInvMasses[i]);
}
```

Solution from the developer:
increment array size



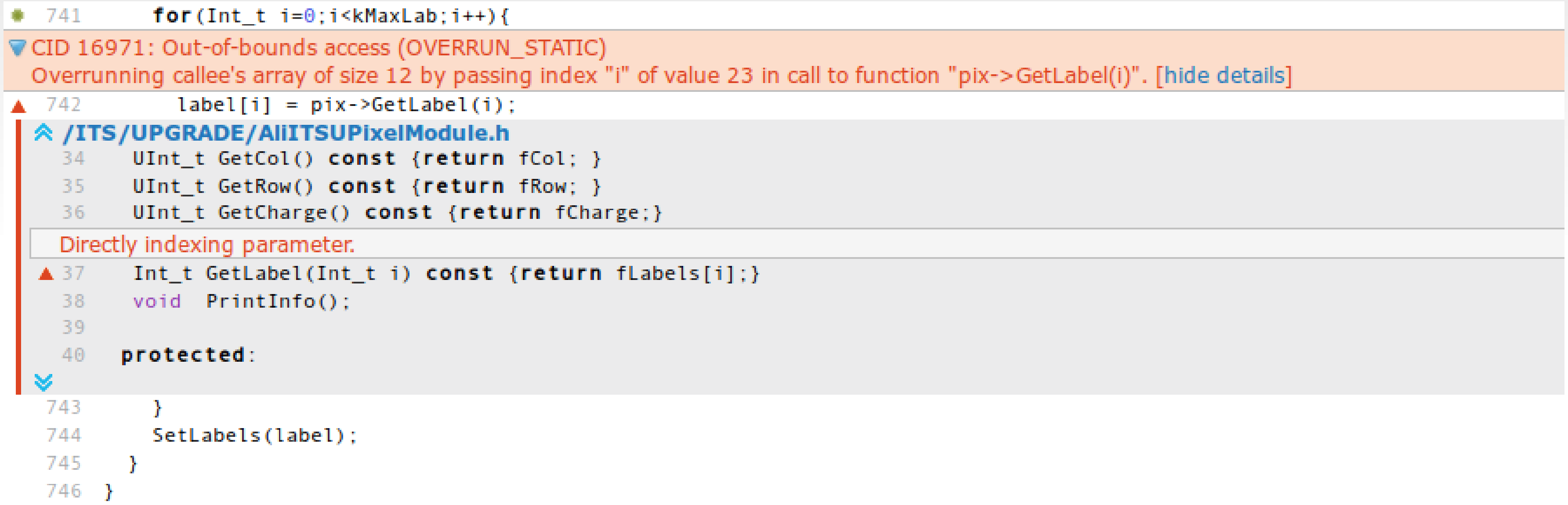
#	Line 102			Line 104		
104	TH2F	*fHPionMggPt;	//!histo for pion mass vs. pT	TH2F	*fHPionMggPt;	//!histo for pion mass vs. pT
105	TH2F	*fHPionMggAsym;	//!histo for pion mass vs. asym	TH2F	*fHPionMggAsym;	//!histo for pion mass vs. asym
106	TH2F	*fHPionMggDgg;	//!histo for pion mass vs. opening angle	TH2F	*fHPionMggDgg;	//!histo for pion mass vs. opening angle
107	TH1F	*fHPionInvMasses[20];	//!histos for invariant mass plots	TH1F	*fHPionInvMasses[21];	//!histos for invariant mass plots

Another example: Out of bounds write [4]

Defect #16971: mismatch in the number of labels within the 'for' loop index and GetLabel() result.

Header:

```
 void  StartEvent();
 enum {kMaxLab=24}; // maximum number of MC labels associated to the cluster
 Int_t ProcessHit(Int_t layer, UInt_t col, UInt_t row, UShort_t charge,Int_t label[kMaxLab]);
```



```
    for(Int_t i=0;i<kMaxLab;i++){
```

CID 16971: Out-of-bounds access (OVERRUN_STATIC)
Overrunning callee's array of size 12 by passing index "i" of value 23 in call to function "pix->GetLabel(i)". [hide details]

```
      label[i] = pix->GetLabel(i);
```

/ITS/UPGRADE/AliITSUPixelModule.h

```
  UInt_t GetCol() const {return fCol; }
  UInt_t GetRow() const {return fRow; }
  UInt_t GetCharge() const {return fCharge;}
```

Directly indexing parameter.

```
  Int_t GetLabel(Int_t i) const {return fLabels[i];}
  void  PrintInfo();

 protected:
```

```
    }
    SetLabels(label);
  }
}
```

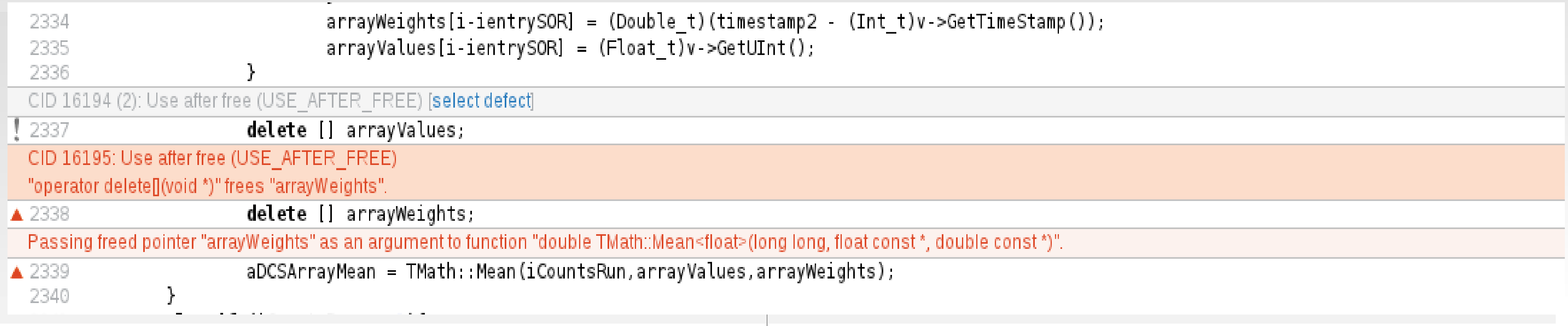
AliITSUPixelModule.h

```
 enum {kMaxLab=12}; // maximum number of MC labels associated to the cluster
  Int_t GetLabel(Int_t i) const {return fLabels[i];}
  void  PrintInfo();

protected:
  UInt_t fCharge;
  UShort_t fModule;
  UInt_t fCol;
  UInt_t fRow;
  Int_t fLabels[kMaxLab];
```

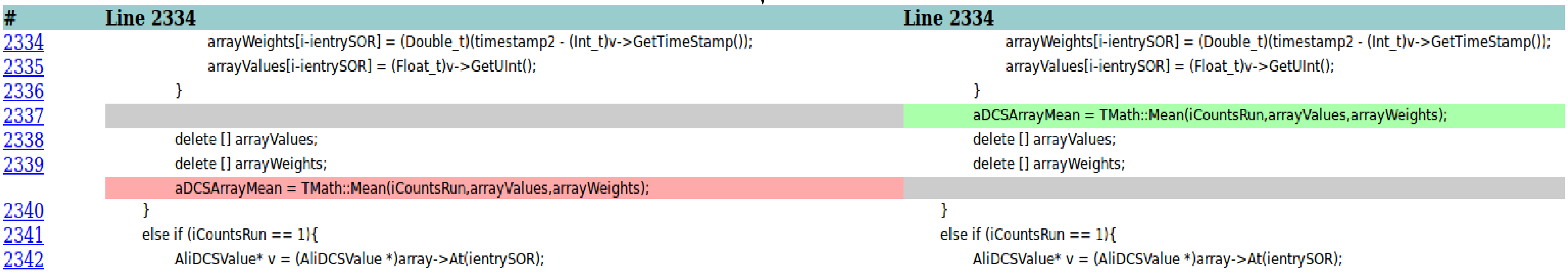
High impact defect: Use after free [5]

Defect #16195: arrayValues and arrayWeights are released in memory, then subsequently used in TMath::Mean() function.



```
                    arrayWeights[i-ientrySOR] = (Double_t)(timestamp2 - (Int_t)v->GetTimeStamp());
                    arrayValues[i-ientrySOR] = (Float_t)v->GetUInt();
            }
CID 16194 (2): Use after free (USE_AFTER_FREE) [select defect]
! 2337          delete [] arrayValues;
CID 16195: Use after free (USE_AFTER_FREE)
"operator delete[](void *)" frees "arrayWeights".
            delete [] arrayWeights;
Passing freed pointer "arrayWeights" as an argument to function "double TMath::Mean<float>(long long, float const *, double const *)".
            aDCSArrayMean = TMath::Mean(iCountsRun,arrayValues,arrayWeights);
        }
```

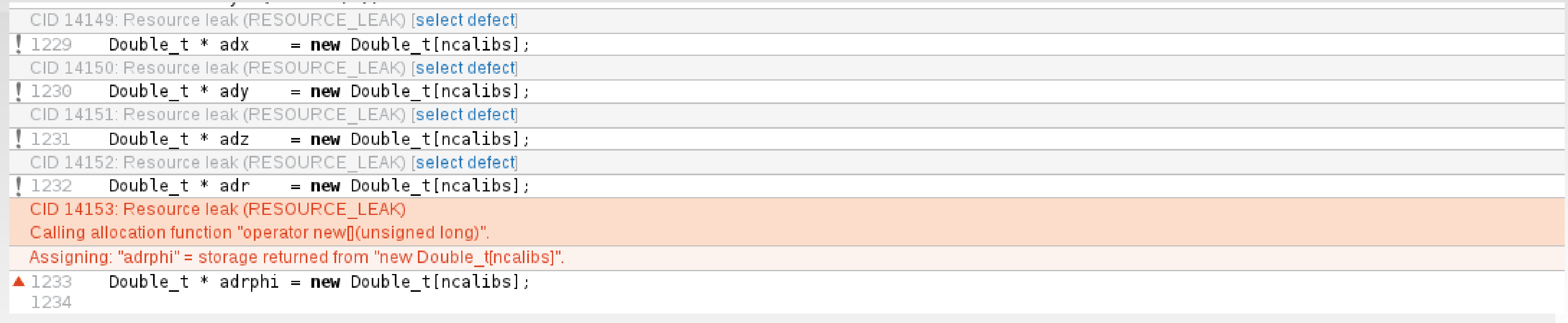
Solution from the developer: get the Mean before deleting the arrays



#	Line 2334	Line 2334
2334	arrayWeights[i-ientrySOR] = (Double_t)(timestamp2 - (Int_t)v->GetTimeStamp());	arrayWeights[i-ientrySOR] = (Double_t)(timestamp2 - (Int_t)v->GetTimeStamp());
2335	arrayValues[i-ientrySOR] = (Float_t)v->GetUInt();	arrayValues[i-ientrySOR] = (Float_t)v->GetUInt();
2336	}	}
2337		aDCSArrayMean = TMath::Mean(iCountsRun,arrayValues,arrayWeights);
2338	delete [] arrayValues;	delete [] arrayValues;
2339	delete [] arrayWeights;	delete [] arrayWeights;
	aDCSArrayMean = TMath::Mean(iCountsRun,arrayValues,arrayWeights);	
2340	}	}
2341	else if (iCountsRun == 1){	else if (iCountsRun == 1){
2342	AliDCSValue* v = (AliDCSValue *)array->At(ientrySOR);	AliDCSValue* v = (AliDCSValue *)array->At(ientrySOR);

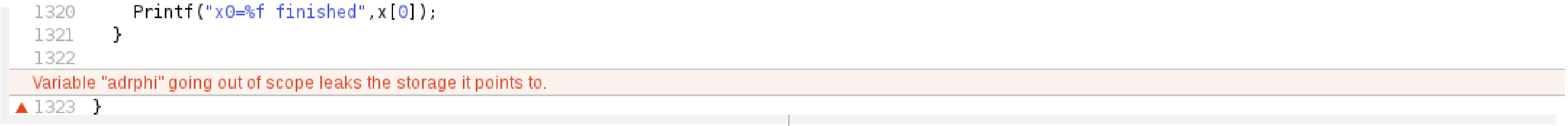
High impact defect: Resource leak [6]

Defect #14153: memory leak occurs when one allocates memory with a **new** operator and does not release the resources at the end of variable scope.



```
CID 14149: Resource leak (RESOURCE_LEAK) [select defect]
   Double_t * adx    = new Double_t[ncalibs];
CID 14150: Resource leak (RESOURCE_LEAK) [select defect]
   Double_t * ady    = new Double_t[ncalibs];
CID 14151: Resource leak (RESOURCE_LEAK) [select defect]
   Double_t * adz    = new Double_t[ncalibs];
CID 14152: Resource leak (RESOURCE_LEAK) [select defect]
   Double_t * adr    = new Double_t[ncalibs];
CID 14153: Resource leak (RESOURCE_LEAK)
Calling allocation function "operator new[](unsigned long)".
Assigning: "adrphi" = storage returned from "new Double_t[ncalibs]".
   Double_t * adrphi = new Double_t[ncalibs];
```

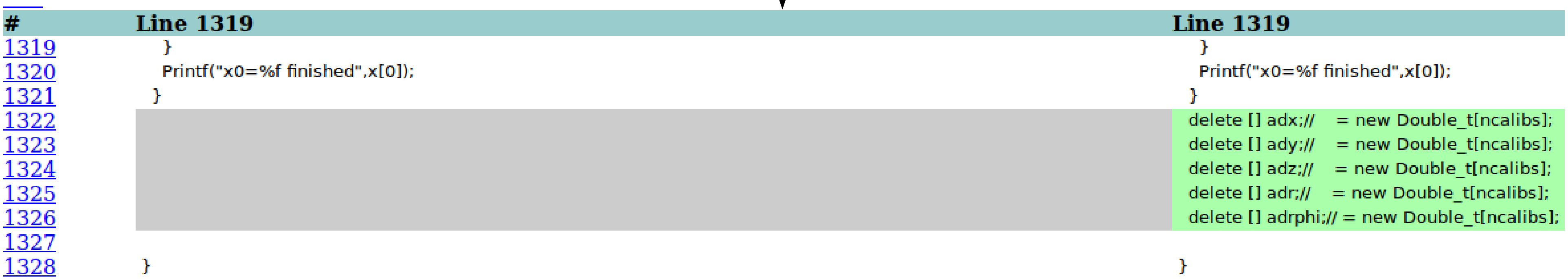
...



```
     Printf("x0=%f finished",x[0]);
   }

Variable "adrphi" going out of scope leaks the storage it points to.
 }
```

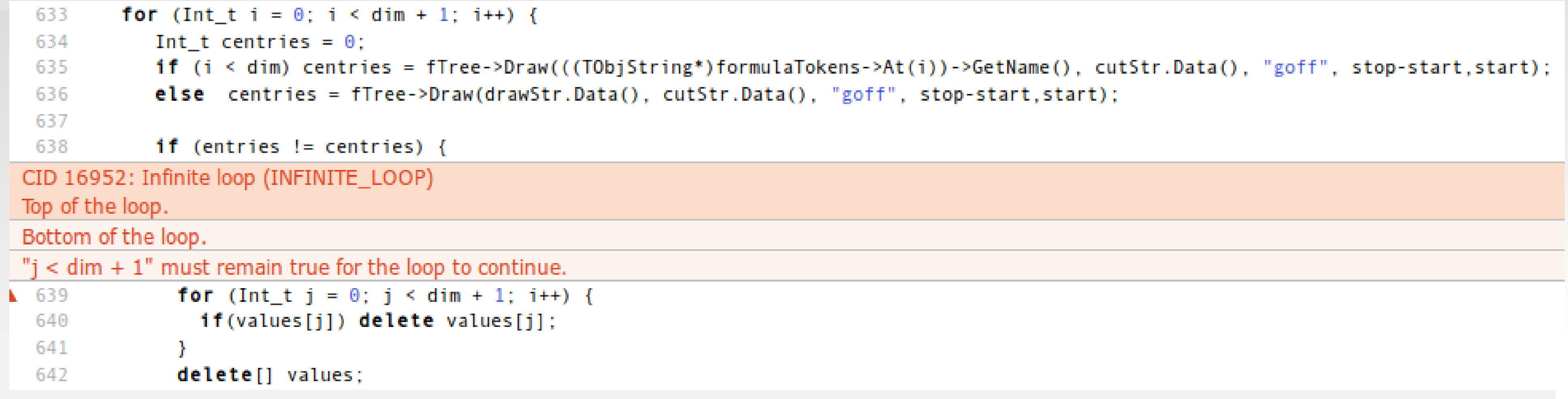
Solution: always use **delete** with the **new** operator



#	Line 1319	Line 1319
1319	}	}
1320	Printf("x0=%f finished",x[0]);	Printf("x0=%f finished",x[0]);
1321	}	}
1322		delete [] adx;// = new Double_t[ncalibs];
1323		delete [] ady;// = new Double_t[ncalibs];
1324		delete [] adz;// = new Double_t[ncalibs];
1325		delete [] adr;// = new Double_t[ncalibs];
1326		delete [] adrphi;// = new Double_t[ncalibs];
1327		
1328	}	}

Medium and Low impact defects

Defect #16952: nothing out of the ordinary?



```
for (Int_t i = 0; i < dim + 1; i++) {
   Int_t centries = 0;
   if (i < dim) centries = fTree->Draw(((TObjString*)formulaTokens->At(i))->GetName(), cutStr.Data(), "goff", stop-start,start);
   else  centries = fTree->Draw(drawStr.Data(), cutStr.Data(), "goff", stop-start,start);

   if (entries != centries) {
```

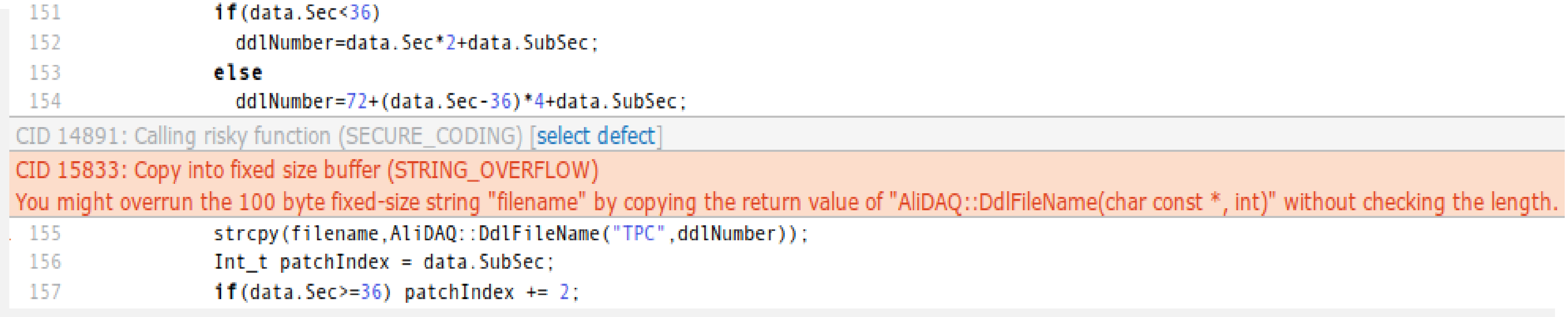
CID 16952: Infinite loop (INFINITE_LOOP)
Top of the loop.

Bottom of the loop.

"j < dim + 1" must remain true for the loop to continue.

```
     for (Int_t j = 0; j < dim + 1; j++) {
       if(values[j]) delete values[j];
     }
     delete[] values;
```

Defect #15833: unsafe copy. [7]



```
if(data.Sec<36)
  ddlNumber=data.Sec*2+data.SubSec;
else
  ddlNumber=72+(data.Sec-36)*4+data.SubSec;
```

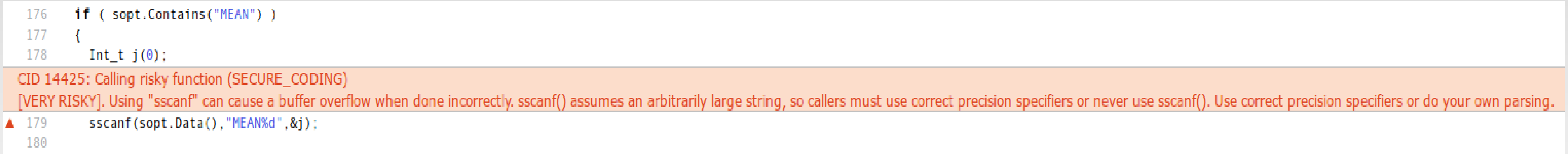
CID 14891: Calling risky function (SECURE_CODING) [select defect]

CID 15833: Copy into fixed size buffer (STRING_OVERFLOW)
You might overrun the 100 byte fixed-size string "filename" by copying the return value of "AliDAQ::DdlFileName(char const *, int)" without checking the length.

```
strcpy(filename,AliDAQ::DdlFileName("TPC",ddlNumber));
Int_t patchIndex = data.SubSec;
if(data.Sec>=36) patchIndex += 2;
```

False positives

Defect #*14425*: only using a string in sscanf and fscanf may be dangerous.

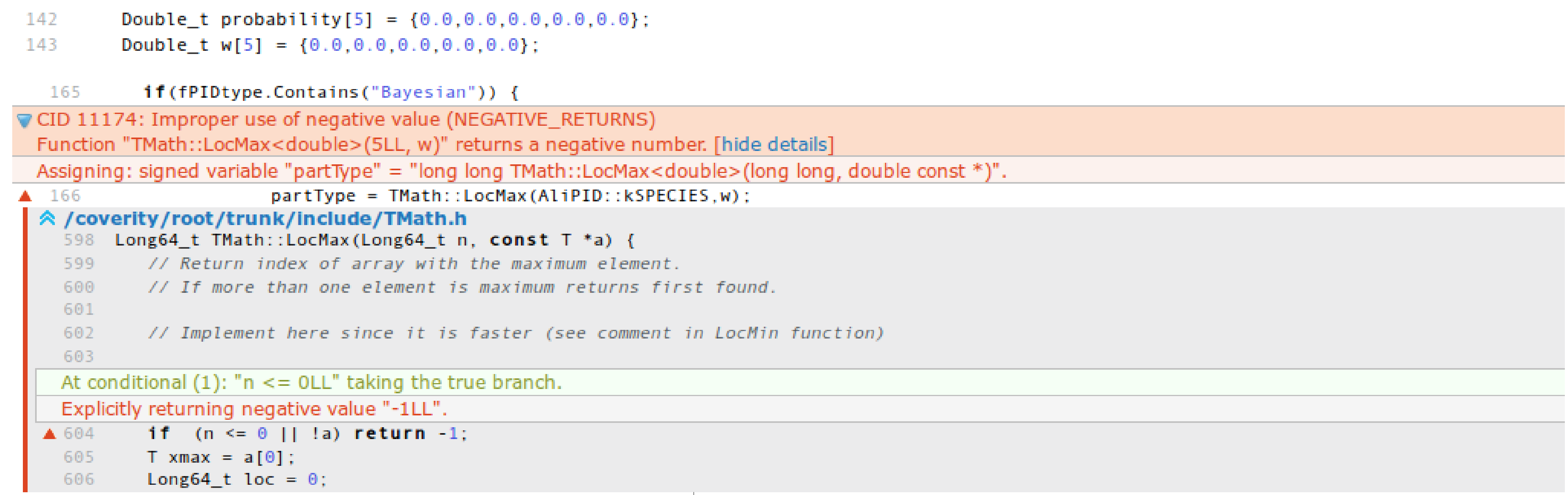


```
  if ( sopt.Contains("MEAN") )
  {
    Int_t j(0);
```

CID 14425: Calling risky function (SECURE_CODING)
[VERY RISKY]. Using "sscanf" can cause a buffer overflow when done incorrectly. sscanf() assumes an arbitrarily large string, so callers must use correct precision specifiers or never use sscanf(). Use correct precision specifiers or do your own parsing.

```
    sscanf(sopt.Data(),"MEAN%d",&j);
```

Defect #*11174*: **kSPECIES** being a constant value can not be less than or equal to 0 and **w** is initialised before use.



```
    Double_t probability[5] = {0.0,0.0,0.0,0.0,0.0};
    Double_t w[5] = {0.0,0.0,0.0,0.0,0.0};

    if(fPIDtype.Contains("Bayesian")) {
```

CID 11174: Improper use of negative value (NEGATIVE_RETURNS)
Function "TMath::LocMax<double>(5LL, w)" returns a negative number. [hide details]

Assigning: signed variable "partType" = "long long TMath::LocMax<double>(long long, double const *)".

```
          partType = TMath::LocMax(AliPID::kSPECIES,w);
```

/coverity/root/trunk/include/TMath.h

```
Long64_t TMath::LocMax(Long64_t n, const T *a) {
   // Return index of array with the maximum element.
   // If more than one element is maximum returns first found.

   // Implement here since it is faster (see comment in LocMin function)

```

At conditional (1): "n <= 0LL" taking the true branch.

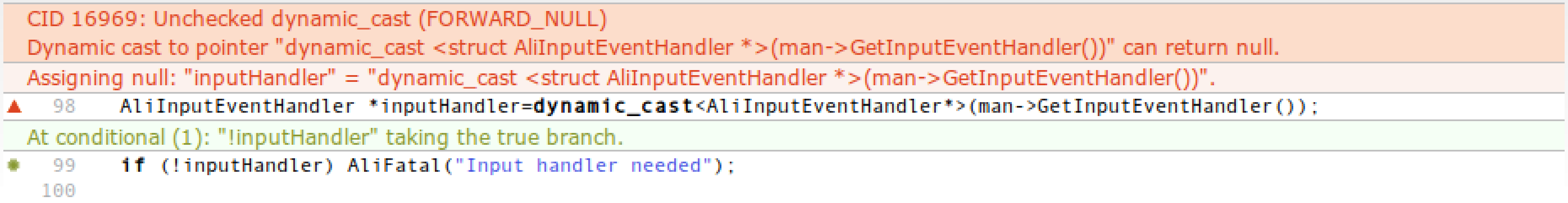
Explicitly returning negative value "-1LL".

```
   if  (n <= 0 || !a) return -1;
   T xmax = a[0];
   Long64_t loc = 0;
```

```
class AliPID : public TObject {
 public:
  enum {
    kSPECIES = 5,     // Number of particle species recognized by the PID
    kSPECIESN = 10,   // Number of charged+neutral particle species recognized by the PHOS/EMCAL PID
    kSPECIESLN = 4    // Number of light nuclei: deuteron, triton, helium-3 and alpha
  };
```

False positives and modeling

Defect #*16969*: defect description suggests that the dynamic_cast may fail and return a NULL. However as seen below the inputHandler will immediately go into AliFatal and exit the program.



CID 16969: Unchecked dynamic_cast (FORWARD_NULL)
Dynamic cast to pointer "dynamic_cast <struct AliInputEventHandler *>(man->GetInputEventHandler())" can return null.

Assigning null: "inputHandler" = "dynamic_cast <struct AliInputEventHandler *>(man->GetInputEventHandler())".

```
AliInputEventHandler *inputHandler=dynamic_cast<AliInputEventHandler*>(man->GetInputEventHandler());
```

At conditional (1): "!inputHandler" taking the true branch.

```
if (!inputHandler) AliFatal("Input handler needed");
```

In order to avoid recurring reports of false positive cases, Coverity static analysis allows for custom function implementations:

- Function producing a false positive case is implemented in a separate .cxx file and adjusted as necessary.
- The source file is then built into a model using cov-make-library.
- Finally, when starting the analysis the –user-model-file is specified along with the library file to use.

Coverity use overview for AliRoot

- The quality of AliRoot code has improved.
- Developers have become more aware of their coding habits. With a centralised system providing complete visibility of all the defects along with some encouragement to fix the problems, developers have become more diligent in their development practice. Defects are now fixed promptly.
- Complementing dynamic analysis, Coverity has greatly helped in debugging hard to diagnose problems.
- Coverity as a static analysis tool used in conjunction with dynamic analysis is an invaluable solution in any development process.

Additional static analysis tools

- **Rule Checker** [8]: performs static code check according to predefined rules, ensuring compliance with both C/C++ coding standards and experiment specific coding conventions. The analysis process is as follows:
  - In the source directory: $ svn update && make check-all
  - After analysis has completed, the following reports are produced and sent to the developer to be fixed:

```
NamingRule: "RN13" : Local variables names start with a lower case letter.
  the variable: AcoHit
  [file: AliACORDEQADataMaker.cxx  line:198] does not start with a lower case letter

CodingRule: "RC11": Make const all member functions that are not supposed to change member
data.
   the method: ASideHasHit
   in file [file:AliFMDOfflineTrigger.cxx line: 60 ] can be declared const
```

The Rule Checker has been developed by the **Bruno Kessler Foundation** [9].

- **cppcheck** [10] – open source static code analysis tool.

Bibliography

1. ‘Welcome to the home page of the ALICE Off-line Project’ <URL http://aliceinfo.cern.ch/Offline> [accessed 26 June 2011] .

2. ‘AliRoot documentation’ <URL http://aliceinfo.cern.ch/Offline/AliRoot/Manual.html> [accessed 26 June 2011] .

3. ‘Test: Code in Development’ <URL http://www.coverity.com/products/> [accessed 26 June 2011] .

4. ‘CWE-119: Improper Restriction of Operations within the Bounds of a Memory Buffer’ <URL http://cwe.mitre.org/data/definitions/119.html> [accessed 26 June 2011] .

5. ‘CWE-416: Use After Free’ <URL http://cwe.mitre.org/data/definitions/416.html> [accessed 26 of June 2011] .

6. ‘CWE-404: Improper Resource Shutdown or Release’ <URL http://cwe.mitre.org/data/definitions/404.html> [accessed 26 June 2011] .

7. ‘CWE-120: Buffer Copy without Checking Size of Input ('Classic Buffer Overflow')’ <URL http://cwe.mitre.org/data/definitions/120.html> [accessed 26 of June 2011] .

8. ‘Coding Conventions’ <URL http://aliceinfo.cern.ch/Offline/AliRoot/Coding-Conventions.html> [accessed 26 June 2011] .

9. ‘Fondazione Bruno Kessler’ <URL http://www.fbk.eu/> [accessed 26 June 2011] .

10. ‘cppcheck: a tool for static C/C+ code analysis’ <URL http://cppcheck.sourceforge.net/> [accessed 26 June 2011] .