Clang Static Analyzer erweitern und nutzen

Was ist der Static Analyzer?

Der Clang Static Analyzer ist in Xcode integriert und Bestandteil des Compilers. Er wird dazu genutzt um den Code "statisch zu analysieren"

Statisch bedeutet der Code wird ohne ausgeführt zu werden auf bestimmte Eigenschaften untersucht.

Was ist der Static Analyzer?

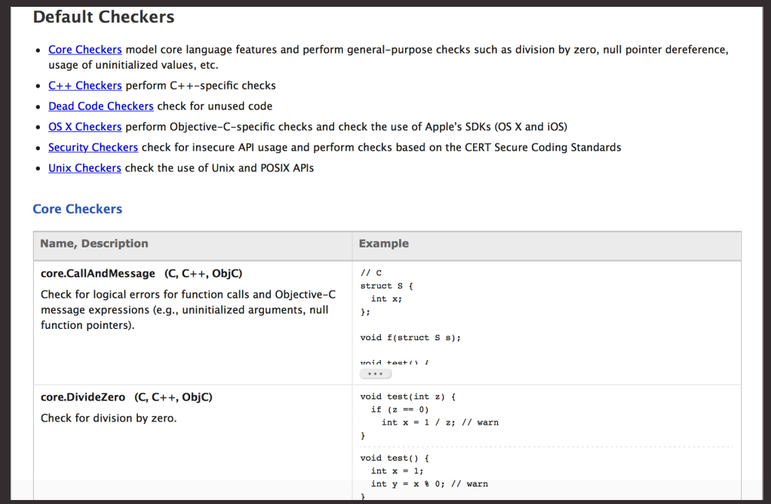
Der Analyzer besteht aus mehreren Checkern, die alle auf verschiedene Eigenschaften prüfen.

Beispiel: Was kann der Analyzer von Haus aus?

- Ungenutzte Variablen finden
- Unerreichbare Code Abschnitt (Goto Fail Bug -> neuere Version von Clang hätte diesen finden müssen)
- Division durch Null feststellen
- Bestimmte andere Eigenschaften (z.B. super Call bei bestimmten Funktionen -> viewDidLoad)

Beispiel: Was kann der Analyzer von Haus aus?

- Vollständige Liste: http://clang-analyzer.llvm.org/available_checks.html



Default Checkers

- Core Checkers model core language features and perform general-purpose checks such as division by zero, null pointer dereference, usage of uninitialized values, etc.
- C++ Checkers perform C++-specific checks
- Dead Code Checkers check for unused code
- OS X Checkers perform Objective-C-specific checks and check the use of Apple's SDKs (OS X and iOS)
- Security Checkers check for insecure API usage and perform checks based on the CERT Secure Coding Standards
- Unix Checkers check the use of Unix and POSIX APIs

Core Checkers

Name, Description	Example

core.CallAndMessage (C, C++, ObjC)
Check for logical errors for function calls and Objective-C message expressions (e.g., uninitialized arguments, null function pointers).
	`// C`
	`struct S {`
	`  int x;`
	`};`
	
	`void f(struct S s);`
	
	`void test() {`
	`...`

core.DivideZero (C, C++, ObjC)
Check for division by zero.
	`void test(int z) {`
	`  if (z == 0)`
	`  int x = 1 / z; // warn`
	`}`
	
	`void test() {`
	`  int x = 1;`
	`  int y = x % 0; // warn`
	`}`

- Xcode Clang ist dem eigentlichen Clang in der Regel einige Versionsnummern hinten dran.
- **Selbst wenn man den Clang nicht erweitern möchte, findet ein neuerer Clang durchaus mehr oder andere Probleme im Code**

Siehe Xcode LLVM/Clang Version:

6.0.1	?	?	-	-	-	-	6.0 (clang-600.0.51) (based on LLVM 3.5svn)[66]
6.1	?	?	-	-	-	-	6.0 (clang-600.0.54) (based on LLVM 3.5svn)[66]
6.1.1	862	241.9	-	-	-	-	6.0 (clang-600.0.56) (based on LLVM 3.5svn)[66]
6.2 beta	862	241.9	-	-	-	-	6.0 (clang-600.0.57) (based on LLVM 3.5svn)[66]
Xcode	**cctools**	**ld64**	**GCC 4.0**	**GCC 4.2**	**LLVM-GCC 4.2**	**LLVM**	**Apple LLVM / Apple Clang**

VS. LLVM Release Schedule:

Upcoming Releases

LLVM 3.6 Release Schedule:

- 14 January 2015: Branch for 3.6 release
- 14 Jan–21 Jan: Testing Phase I
- 22 Jan–29 Jan: Fix bugs from Testing Phase I
- 30 Jan–6 Feb: Testing Phase II
- 7 Feb–14 Feb: Fix bugs from Testing Phase II
- 21 February: 3.6.0 RELEASE!

How to build your own Clang

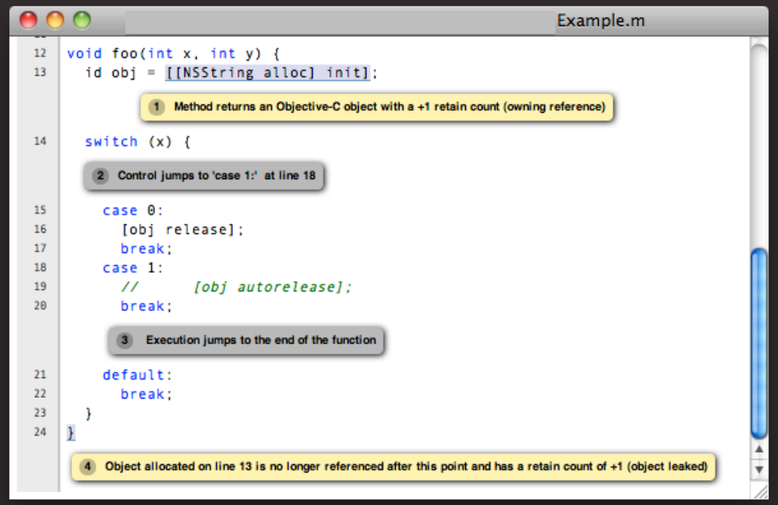
1. LLVM & Clang auschecken. Siehe http://llvm.org/docs/GettingStarted.html#getting-started-quickly-a-summary
2. Build-Ordner erstellen
3. Im Ordner Terminal öffnen und `pathToLLVM/configure` ausführen
4. Im Terminal: `make -jN`
   wobei `N` = Anzahl Prozessor + 1 (wesentlich schneller)

Eigenen Clang-Analyzer benutzen

Zwei Möglichkeiten:

- Konsolenoutput für eine Datei:
  `yourClang -cc1 yourOBJCFile.m`

- ScanBuild HTML Ouput (mit ScanView): (Im Projektordner)
  `pathToScanBuild xcodebuild`



(Hinweis: `scanBuild` muss sich im Überordner relativ zum `bin`-Ordner des kompilierten Clang befinden)

Eigenen Clang-Analyzer benutzen

Den aktuellsten Build findet man auch fertig kompiliert unter:
http://clang-analyzer.llvm.org

Eigenen Clang-Analyzer in Xcode einbinden

- In den Clang Quellen existiert Tool namens: `set-xcode-analyzer`
- ScanBuild muss wie oben beschrieben im übergeordneten Ordner zum Clang-bin-Ordner platziert sein
- Eignen Clang einstellen:

```
sudo ./set-xcode-analyzer --use-checker-build=PATH_TO_CLANGPARENTFOLDER_WITH_SCANBUILD
```

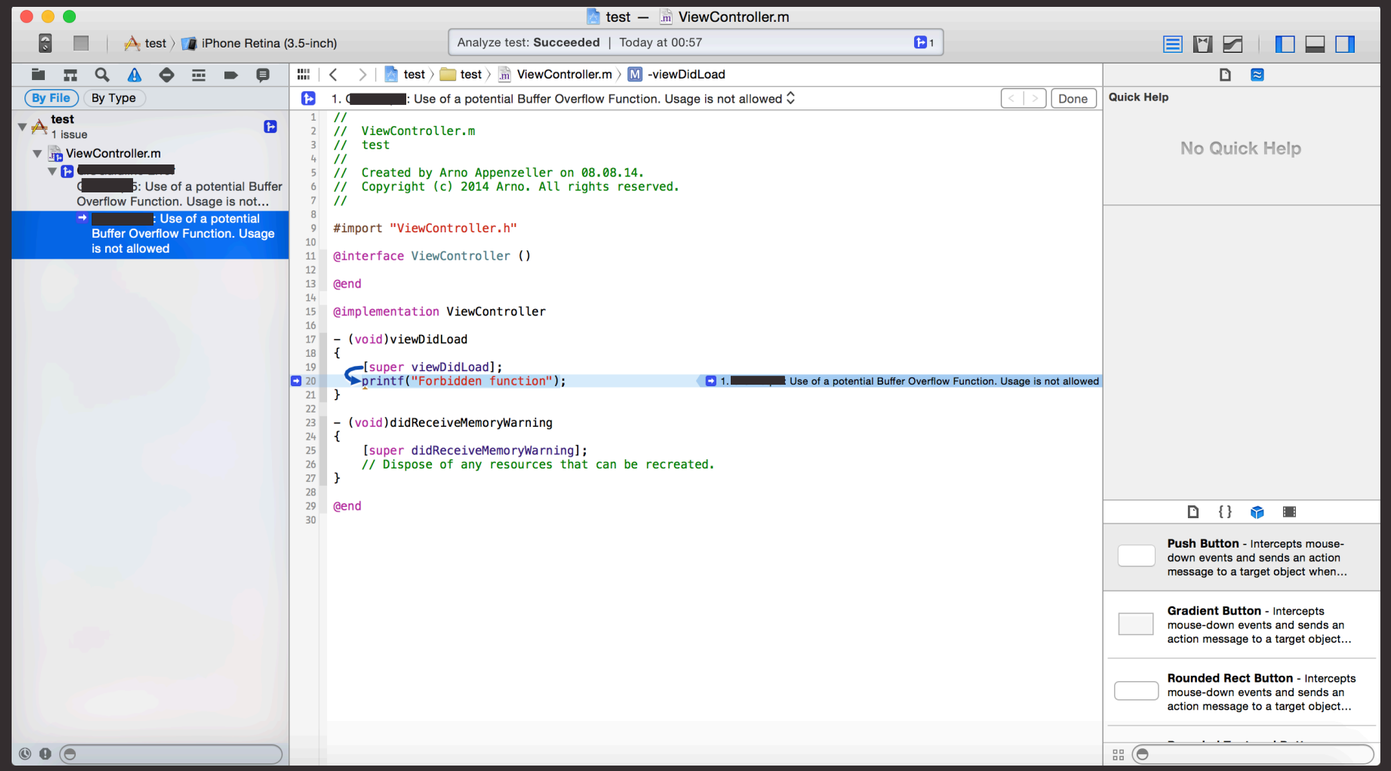
Eigenen Clang-Analyzer in Xcode einbinden

- Xcode Clang zurücksetzen:

```
sudo ./set-xcode-analyzer --use-xcode-clang
```

- Funktioniert nun über Analyze-Button

Eigenen Clang-Analyzer in Xcode einbinden



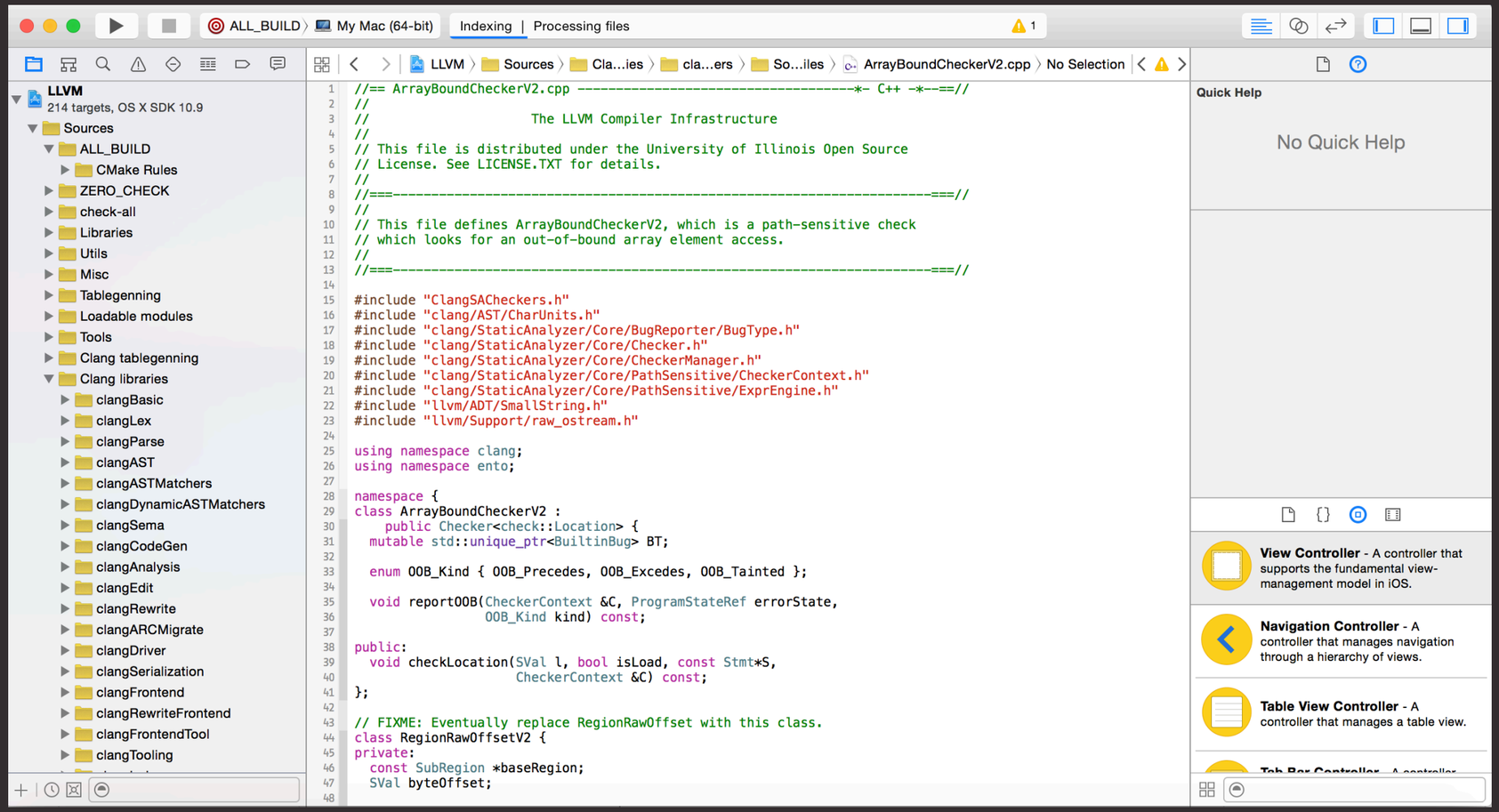
Clang erweitern - Warum?

- Umsetzung von (Unternehmens)-Richtlinien (Verbotene Funktionen, etc.)
- Verifizierung von Framework Verwendung
- Coding Styles
- Sicherheitserweiterungen, etc.
- (Außerdem ist es recht spannend 😜)

Clang erweitern - Start

- Man kann aus der Clang Source ein Xcodeproject erstellen
- wesentlich angenehmer zu entwicklen als mit Editor und Terminal
- `cmake -G Xcode -DCMAKE_BUILD_TYPE=Debug pathToLLVMSource`

Clang erweitern - Start




```
//== ArrayBoundCheckerV2.cpp ------------------------------------*- C++ -*--==//
//
//                     The LLVM Compiler Infrastructure
//
// This file is distributed under the University of Illinois Open Source
// License. See LICENSE.TXT for details.
//
//===----------------------------------------------------------------------===//
//
// This file defines ArrayBoundCheckerV2, which is a path-sensitive check
// which looks for an out-of-bound array element access.
//
//===----------------------------------------------------------------------===//

#include "ClangSACheckers.h"
#include "clang/AST/CharUnits.h"
#include "clang/StaticAnalyzer/Core/BugReporter/BugType.h"
#include "clang/StaticAnalyzer/Core/Checker.h"
#include "clang/StaticAnalyzer/Core/CheckerManager.h"
#include "clang/StaticAnalyzer/Core/PathSensitive/CheckerContext.h"
#include "clang/StaticAnalyzer/Core/PathSensitive/ExprEngine.h"
#include "llvm/ADT/SmallString.h"
#include "llvm/Support/raw_ostream.h"

using namespace clang;
using namespace ento;

namespace {
class ArrayBoundCheckerV2 :
    public Checker<check::Location> {
  mutable std::unique_ptr<BuiltinBug> BT;

  enum OOB_Kind { OOB_Precedes, OOB_Excedes, OOB_Tainted };

  void reportOOB(CheckerContext &C, ProgramStateRef errorState,
                 OOB_Kind kind) const;

public:
  void checkLocation(SVal l, bool isLoad, const Stmt*S,
                     CheckerContext &C) const;
};

// FIXME: Eventually replace RegionRawOffset with this class.
class RegionRawOffsetV2 {
private:
  const SubRegion *baseRegion;
  SVal byteOffset;
```

Checker Plugin entwickeln - Basics

- Bestehende Checker befinden sich in `tools/clang/lib/StaticAnalyzer/Checkers`
- Checker, wie der gesamte LLVM/Clang, werden in C++ entwickelt
- Für einen neuen Checker einfach eine neue Datei im Checkers Ordner erstellen.

Checker Plugin entwickeln - Basics

- Datei muss in der *CMakeLists.txt* im Ordner hinzugefügt werden
- Checker muss in *Checkers.td* registriert werden (später mehr)

Checker Plugin entwickeln - Einfacher eigener Checker

Wir möchten auf C-Funktionsaufrufe wie *printf*,*strcat*,*strcpy*,*strncat* finden und davor warnen

(Neuere Xcode Version haben dies bereits als Option für Warnings)

Checker Plugin entwickeln - Aufbau Checker I

- Ein Checker prüft auf bestimmte Events (z.B. `checkPreCall` das Statements vor ihrem "Aufruf" prüft -> Standardevent)
- Weitere Events: `checkPostCall`,`checkPreObjCMessage`,`checkEndFunction`, `checkDeadSymbols`,etc.
- Ein Event übergibt immer den aktuelle `CheckerContext`, der z.B. aktuellen CalleeIdentifier oder ASTContext enthält.

Checker Plugin entwickeln - Aufbau Checker II

- Der aktuelle Callee wird als IdentifierInfo gespeichert, wir suchen in unserem Fall etwas wie *printf*
- *printf* ist nicht immer *printf* - wir müssen den passenden Identifier anhand des aktuellen AST initiieren:

```
IIprintf = &ASTContext.Idents.get("printf");
```

Checker Plugin entwickeln - Aufbau Checker III

- Nun können wir im Event prüfen, ob der Identifier des aktuellen Context *printf* ist:

```
void CocoheadSecurityFunctionChecker::checkPreCall(const CallEvent &Call, CheckerContext &C) const {
  initIdentifierInfo(C.getASTContext()); //initiiert die Identifier
  const IdentifierInfo *ID = Call.getCalleeIdentifier();

  //check if printf
  if ( ID == IIprintf){
  //throw bug
  }
}
```

Checker Plugin entwickeln - Aufbau Checker IV

- Bugs erstellt man indem man sie zuerst Global definiert, via reset

```
boBug.reset(new BugType(this, "Potential BufferOverflow Function", "CocoheadSecurityFunctionChecker Error"));
```

- Man wirft sie aus indem man dem Context einen Bugreport mit Bug-Art,Beschreibung und Location übergibt

```
ExplodedNode *ErrNode = C.generateSink(); //Location anhand des Context (Hinweis: Sink beendet alle weiteren Checks anhand dieses Ablaufs)
BugReport *bug = new BugReport(*ufhBug, "CocoheadSecurityFunctionChecker: Use of a unsafe file handeling function. Usage is not allowed.", ErrNode);
C.emitReport(bug); //auslösen des reports
```

Checker Plugin entwickeln - Checker registrieren I

Ein Checker muss dem CheckerManager mitteilen, dass es ihn gibt:

```
void ento::registerCocoheadSecurityFunctionChecker(CheckerManager &mgr) {
  mgr.registerChecker<CocoheadSecurityFunctionChecker>();
}
```

Checker Plugin entwickeln - Checker registrieren II

- In *Checkers.td* sind alle Checker aufgelistet und in bestimmte Kategorien eingeteilt
- Um entwickelt Checker hinzuzufügen folgenden Code einfügen (in bestehende Kategorien - wir wählen *Security*)

```
let ParentPackage = Security in {
//a lot of other code
def CocoheadSecurityFunctionChecker :Checker<"CocoheadSecurityFunctionChecker">,
  HelpText<"Checks for security">,
  DescFile<"CocoheadSecurityFunctionChecker.cpp">;
 }
```

Checker Plugin entwickeln - Fertigstellung

Konfiguration ALL_BUILD bauen:

Neuer Clang befindet sich nun in: "Debug/bin" (je nach Konfiguration)

(Kompletter Code hier: github.com/arnoappenzeller)

Checker Plugin entwickeln - Fazit

- Anhand dieser Grundlage lässt sich bereits ein einfacher eigner Checker erstellen.
- Für Objective-C ist das ganze etwas komplizierter, da hier mit Nachrichten und Selectors gearbeitet wird.
- Ein komplexeres Beispiel ist auf Github (prüft ob ein JailbreakCheck durchgeführt wird, wenn ein bestimmter Parameter gesetzt ist)

Links

1. Clang Analyzer Seite
2. Checker Developer Manual
3. Clang Doxygen (Für komplexere Entwicklungen notwendig!)
4. "Building a Checker in 24 hours" (Talk von LLVM Dev Conference) Slides Video

Fragen/Kontakt?

- Per Twitter unter @arno_app
- Github: github.com/arnoappenzeller
- Mehr über mich und meine iOS Projekte: app-enzeller.com