Masters Capstone Project - Type Annotation Analysis Using the .NET Compiler Platform
Student: Theodore Sill
Advisor: Professor Matthew Fluet

This file contains the contents of the Git repository created to manage the source code written for my capstone project.

CSharpQual

This solution contains a diagnostic analyzer and code fix created as part of milestone one. The primary goal was to become more familiar with the .NET Compiler Platform ("Roslyn") by addressing a common programming error. When formatting a string in C# (prior to C# 6.0 which introduces string interpolation) a format string is used which contains replacement tokens. If the number of additional arguments provided doesn't match the number of tokens to replace then a runtime error can occur. This project searches for occurrences of this mismatch and can correct the issue automatically.

EncryptedSandbox

This solution contains a test application which may be loaded in the Visual Studio (VS) experimental hive instance executing SharpChecker. It contains a variety of expressions and statements which exercise "Encrypted" attribute verification. The examples are primarily invocation expressions and assignment statements.

InheritedSandbox

This solution contains another test application used to verify the rules associated with overriding methods. Here we are also verifying "Encrypted" attribute application. The examples are a variety of method declarations.

(Content below added or modified after milestone two)

TaintedSandbox

This solution contains another test application used to verify the "tainted" type system. Here we are verifying the "Tainted" and "Untainted" attributes. This type system takes into consideration the hierarchy of these attributes, so that "Untainted" may be used where "Tainted" is expected. That is to say we allow for subtyping among type qualifiers. This also served as a sample for adding a new type system to SharpChecker, and the associated steps will appear in the user guide.

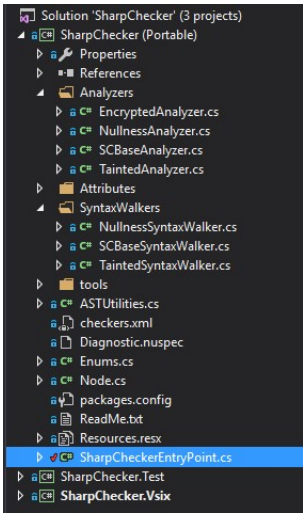
NullnessSandbox

This solution contains another test application used to verify the "nullness" type system. Here we are verifying the "NonNull" and "MaybeNull" attributes. This type system is concerned with unsafely dereferencing an identifier, and will present an error when a "MaybeNull" variable is dereferenced. This type system also introduces the concept of context sensitivity by looking for explicit comparisons to null. When a code block is guarded by an explicit check for null the annotated type of the identifier which is explicitly compared to null is refining to "NonNull" within that block.

PosterSandbox

This solution contains another test application which was used to generate sample images for the poster presentation. The poster has limited space, so in some cases it made sense to format things differently then one normally would, to make the best use of the space available.

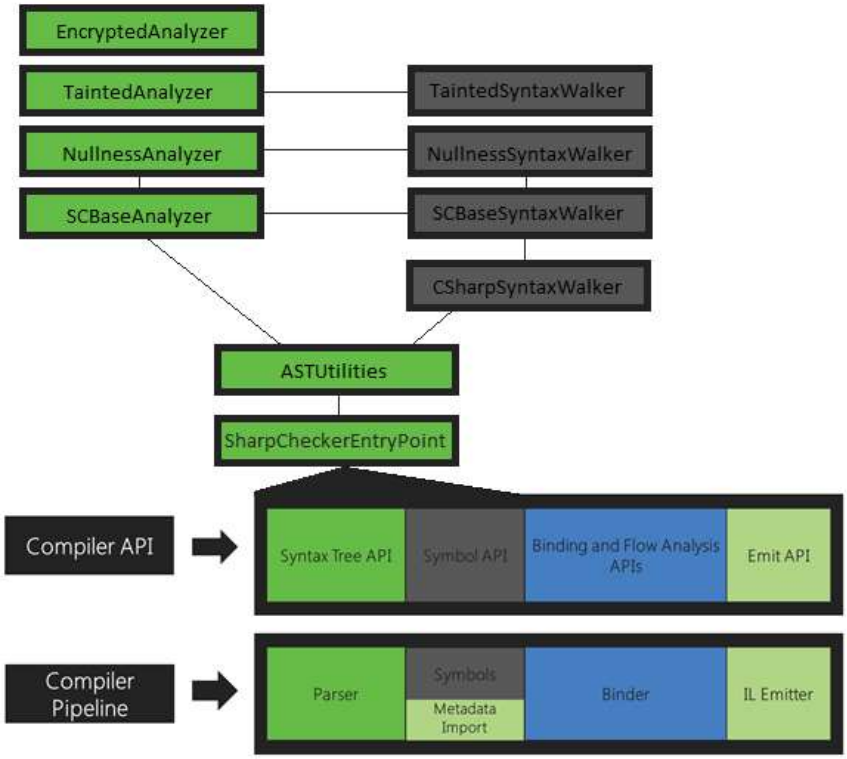
SharpChecker



This solution (pictured above) contains the SharpChecker framework, which is the primary deliverable. Its purpose is to verify that target code respects the rules of an annotated type system. In its current state it supports three such systems: Encrypted, Nullness, and Tainted. It was designed to be extensible, so that it may be instantiated to new type systems and modified only as much as is necessary to support the unique characteristics of the new type system. The following is a description of the most important classes:

- SharpCheckerEntryPoint – This is the entry point for SharpChecker. It is here that we register with the .NET Compiler Platform by extending the DiagnosticAnalyzer class and decorating our class with the [DiagnosticAnalyzer(LanguageNames.CSharp)] attribute. In the “Initialize” method we instantiate our ASTUtilites object (described below), and register actions to be performed when steps in the analysis life-cycle occur.
- ASTUtilities – This class manages the collection of diagnostic rules and analyzers, and provides some functionality which is common to Analyzers and SyntaxWalkers.
- Analyzers – This folder contains SCBaseAnalyzer and those classes which extend it. These classes are concerned with collecting the type annotations, which occur in source code as attributes. The annotations are stored in a symbol dictionary where the key is a syntax node, and the value is a collection of attribute names.
- SyntaxWalkers – The classes in this folder extend SCBaseSyntaxWalker which in turn extends the CSharpSyntaxWalker class. This allows us to build upon an implementation of the visitor design pattern provided by the .NET Compiler Platform. Using this mechanism we visit all of the syntax nodes which we are concerned with as we walk over the AST and verify that the appropriate effective type annotation is present. When not present, we register a diagnostic as a side effect, and this diagnostic is presented to the user in the Visual Studio IDE.

The figure below presents an overview of the SharpChecker framework, and the relationship between the classes. As you can see the NullnessAnalyzer extends SCBaseAnalyzer. Similarly, each of those classes pictured above NullnessAnalyzer also extends SCBaseAnalyzer. The EncryptedAnalyzer was simple enough that it did not need to create a specialized SyntaxWalker class, but instead inherited the base functionality present in SCBaseSyntaxWalker. This includes the enforcement of data flow and subtyping in assignments and pseudo-assignments (such as when providing an argument to a method invocation or returning a value).



The Compiler API and Compiler Pipeline sections pictured above are available at https://github.com/dotnet/roslyn/wiki/Roslyn-Overview

localnuget

This directory was established as a nuget source for the SharpChecker project. This allows one to publish stable versions of the framework to this source, and upgrade the versions in use in SharpChecker itself, and other target solutions with the “Manage NuGet Packages” functionality which is built into Visual Studio.

Experimental Results

This spreadsheet contains the quantifiable observations from applying SharpChecker to target code.