

18732: Secure Software Systems

Homework 2B Discussion

Michelle Mazurek

October 13, 2010

Administrivia

- Due Friday 10/22
- Get your VM from Michelle, Lujo, or Anupam
 - During office hours
 - By request
- Read the reference documents, headers before you ask questions about Coverity-isms

Fun with Coverity syntax

- Use the checker reference to understand what the defect reports are telling you
- Use the Extend reference to understand how to use the SDK
 - Also the sample checkers
 - You may have to read the Extend header files
- Instructions point you at the documents

Coverity analysis in 3 steps

1. Build the checker

Details in the instructions!

- SA: Many pre-built checkers
- Use **build-checker** to build your own

2. Translate the target source to intermediate

- Use **cov-build** with **make**
- Use **cov-translate** for a single source file
- Use **--dir** to specify where to put intermediate

3. Run the analyzer

- Use **cov-analyze** for pre-built SA
- Use **./<mychecker>** for yours
- Use **--dir** to specify the target intermediate

Anatomy of a defect report

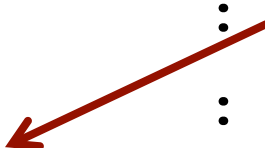
Analysis summary report:

```
Files analyzed           : 1
Total LoC input         : 11751
Functions analyzed      : 8
Paths analyzed          : 1162
New defects found       : 6 Total
                        : 2 DEADCODE
                        : 3 SECURE_CODING
                        : 1 STRING_OVERFLOW
```

Anatomy of a defect report

Analysis summary report:

```
Files analyzed           : 1
Total LoC input         : 11751
Functions analyzed      : 8
Paths analyzed          : 1162
New defects found       : 6 Total
                        : 2 DEADCODE
                        : 3 SECURE_CODING
./<ad>/c/output/DEADCODE.ERRORS.XML 7ERFLOW
```



Anatomy of a Checker

```
#include "extend-lang.hpp"
START_EXTEND_CHECKER( ref_arg, simple );

ANALYZE_TREE()
{
    Arg arg;

    if( MATCH( arg ) ) {
        ReferenceType ref_type;
        // formal_type may be 0 (for arguments passed to ...)
        const types::type_t *formal_type = arg.get_formal_type();
        if(formal_type && MATCH_TREE( ref_type, formal_type ) ) {
            OUTPUT_ERROR( arg << " is a reference arg of type " << ref_type );
        }
    }
}

END_EXTEND_CHECKER();

MAKE_MAIN( ref_arg )|
```

← Insert globals, helper functions here.

Anatomy of a Checker

```
#include "extend-lang.hpp"

START_EXTEND_CHECKER( ref_arg, simple ),
ANALYZE_TREE()
{
    Arg arg;

    if( MATCH( arg ) ) {
        ReferenceType ref_type;
        // formal_type may be 0 (for arguments passed to ...)
        const types::type_t *formal_type = arg.get_formal_type();
        if( formal_type && MATCH_TREE( ref_type, formal_type ) ) {
            OUTPUT_ERROR( arg << " is a reference arg of type " << ref_type );
        }
    }
}

END_EXTEND_CHECKER();

MAKE_MAIN( ref_arg |
```

Name of checker

- ref_arg.cpp
- ref_arg.errors.xml

Anatomy of a Checker

```

#include "extend-lang.hpp"

START_EXTEND_CHECKER( ref_arg, simple );

ANALYZE_TREE()
{
  Arg arg;
  if( MATCH( arg ) ) {
    ReferenceType ref_type;
    // formal_type may be 0 (for arguments passed to ...)
    const types::type_t *formal_type = arg.get_formal_type();
    if(formal_type && MATCH_TREE( ref_type, formal_type ) ) {
      OUTPUT_ERROR( arg << " is a reference arg of type " << ref_type );
    }
  }
}

END_EXTEND_CHECKER();

MAKE_MAIN( ref_arg )|

```

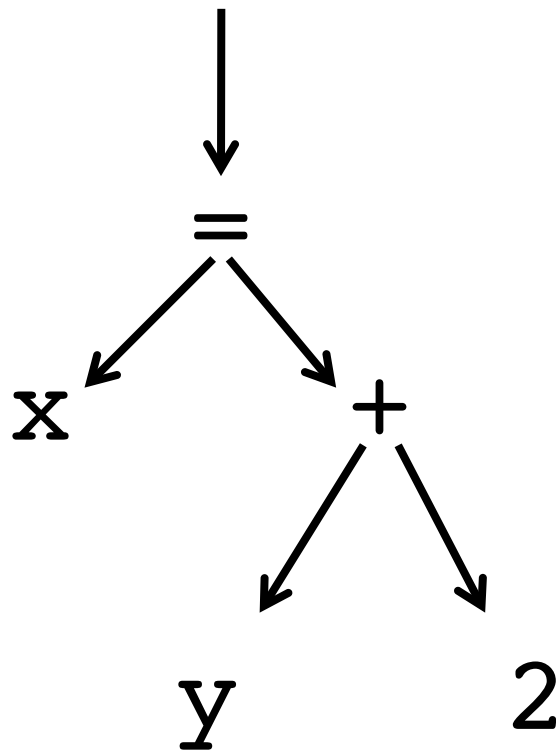
Main matching function

**Declare patterns to match against
(see patterns in docs, headers)**

**Arg matches a formal argument
(Fun, LocalVar, Pointer, ArrayIndex, etc.)
(Integer is a pattern, int is a variable)**

Abstract syntax tree

$x = y + 2$ (statement)



Visit order:

y
 2
 $y + 2$
 $x = y + 2$ (assignment)
 x
 $x = y + 2$ (statement)

Anatomy of a Checker

```

#include "extend-lang.hpp"

START_EXTEND_CHECKER( ref_arg, simple );

ANALYZE_TREE()
{
    Arg arg;
    Match current tree node
    if( MATCH( arg )) { (now arg holds the current node)
        ReferenceType ref_type;
        // formal_type may be 0 (for arguments passed to ...)
        const types::type_t *formal_type = arg.get_formal_type();
        if(formal_type && MATCH_TREE( ref_type, formal_type ) ) {
            OUTPUT_ERROR( arg << " is a reference arg of type " << ref_type );
        }
    }
}

END_EXTEND_CHECKER();

MAKE_MAIN( ref_arg )|

```

Anatomy of a Checker

```

#include "extend-lang.hpp"

START_EXTEND_CHECKER( ref_arg, simple );

ANALYZE_TREE()
{
    Arg arg;

    if( MATCH( arg ) ) {
        ReferenceType ref_type;
        // formal_type may be 0 (for arguments passed to...)
        const types::type_t *formal_type = arg.get_formal_type();
        if(formal_type && MATCH_TREE( ref_type, formal_type ) ) {
            OUTPUT_ERROR( arg << " is a reference arg of type " << ref_type );
        }
    }
}

END_EXTEND_CHECKER();

MAKE_MAIN( ref_arg )|

```

Type is a property of a pattern
(See types in docs, headers)

Check for type "reference"

Anatomy of a Checker

```

#include "extend-lang.hpp"

START_EXTEND_CHECKER( ref_arg, simple );

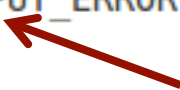
ANALYZE_TREE()
{
    Arg arg;

    if( MATCH( arg ) ) {
        ReferenceType ref_type;
        // formal_type may be 0 (for arguments passed to ...)
        const types::type_t *formal_type = arg.get_formal_type();
        if(formal_type && MATCH_TREE( ref_type, formal_type ) ) {
            OUTPUT_ERROR( arg << " is a reference arg of type " << ref_type );
        }
    }
}

END_EXTEND_CHECKER();

MAKE_MAIN( ref_arg )|

```


Generates an entry in summary.txt, ref_arg.errors.xml, with descriptive text. (You can also use cout to debug print.)

Anatomy of a Checker

```
#include "extend-lang.hpp"

START_EXTEND_CHECKER( ref_arg, simple );

ANALYZE_TREE()
{
    Arg arg;

    if( MATCH( arg ) ) {
        ReferenceType ref_type;
        // formal_type may be 0 (for arguments passed to ...)
        const types::type_t *formal_type = arg.get_formal_type();
        if(formal_type && MATCH_TREE( ref_type, formal_type ) ) {
            OUTPUT_ERROR( arg << " is a reference arg of type " << ref_type );
        }
    }
}

END_EXTEND_CHECKER();
MAKE_MAIN( ref_arg )
```

Required closing stuff
(checker name again)