Concurrent Technologies Corporation

Gaps in Static Analysis Tool Capabilities

Providing World-Class Services for World-Class Competitiveness

Concurrent Technologies Corporation

Carnegie Mellon CyLab

Overview

- Gaps in Static Analysis tools as identified during the evaluation of five (5) commercially available static analysis tools
  - Collaborative effort between *CTC,* Carnegie Mellon CyLab
- Topics
  - Perspective
  - Desirable Characteristics
  - Significant Capability Gaps
  - A Way Forward
  - Summary

Perspective: Experience with the Tools

- Thoroughly examined and compared five (5) commercial static analysis tools
  - Subjected to a variety of tests involving both synthetic and “real-world” code
- Helping to develop a methodology for extracting assurance about software using static analysis

Perspective: Constraints

- Assume two (2) third party analysts (possibly experienced)
- Assume one month to complete the evaluation
- Assume projects may scale to millions of lines of code (LOC)
- Assume no single tool covers requirements
- Need tools that can facilitate this type of analysis
  - Worst possible evaluation scenario
    - No knowledge of the code
    - Varying degrees of expertise
    - Different requirements than the developers

Desirable Characteristics

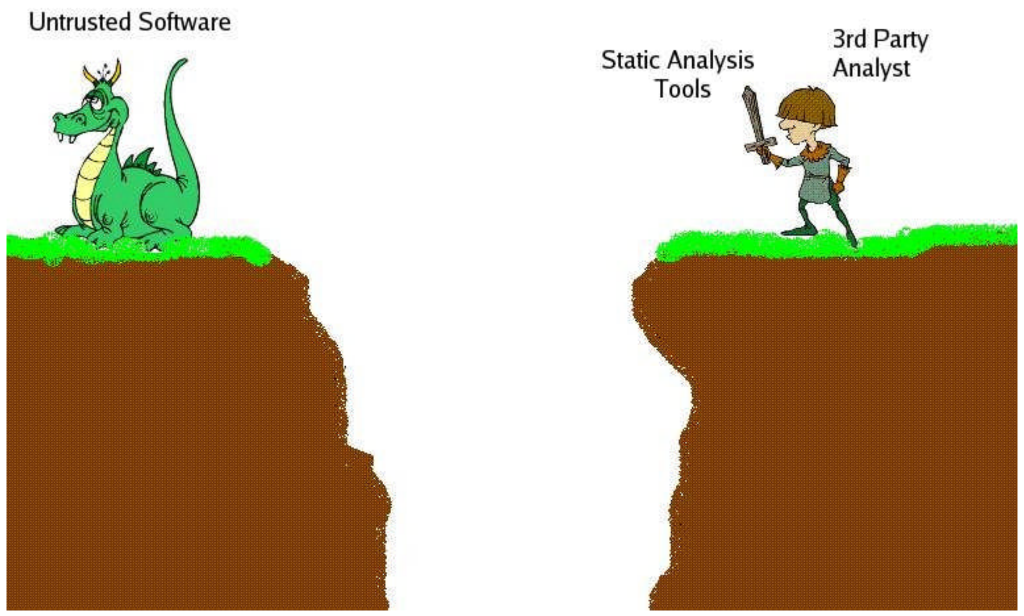
- Characteristics that still cut across all tools:
  - Low False Positive (FP) and False Negative (FN) rates
  - Deep coverage of at least one flaw type
  - Wider coverage of major security flaw types
  - Easy to perform analysis
  - Support for major programming languages
  - Tunable

Existing Features (the good)

- Code understanding facilities
- Extensibility Application Programming Interfaces (API)
- Interprocedural analysis
- Review/Audit Support
- API modeling
- Flow analysis
- Flaw pre/post conditions
- Security knowledge bases
- Data export & import
- Code browsing
- Trend analysis
- Scalability
- Multi-language analysis
- Analysis without build modification
- Flaw Filtering

Significant Gaps

- Reliability
  - Is the tool's analysis thorough enough?
- Transparency
  - Can we tell what the tool is and isn't doing?
- Flaw Detection
  - Can tool that can detect flaws of a certain type?
- Interoperability
  - Can we get multiple tools to work together?



Reliability

- Really Hard problems:
  - Handling in-line assembly
  - Pointer arithmetic
  - Code generated or loaded at run time
  - Deep Data Flow (DF) – Control Flow (CR) – Affine Relations (AR) – Value Set (VS) – Context Sensitive (CS) analysis (all at once)

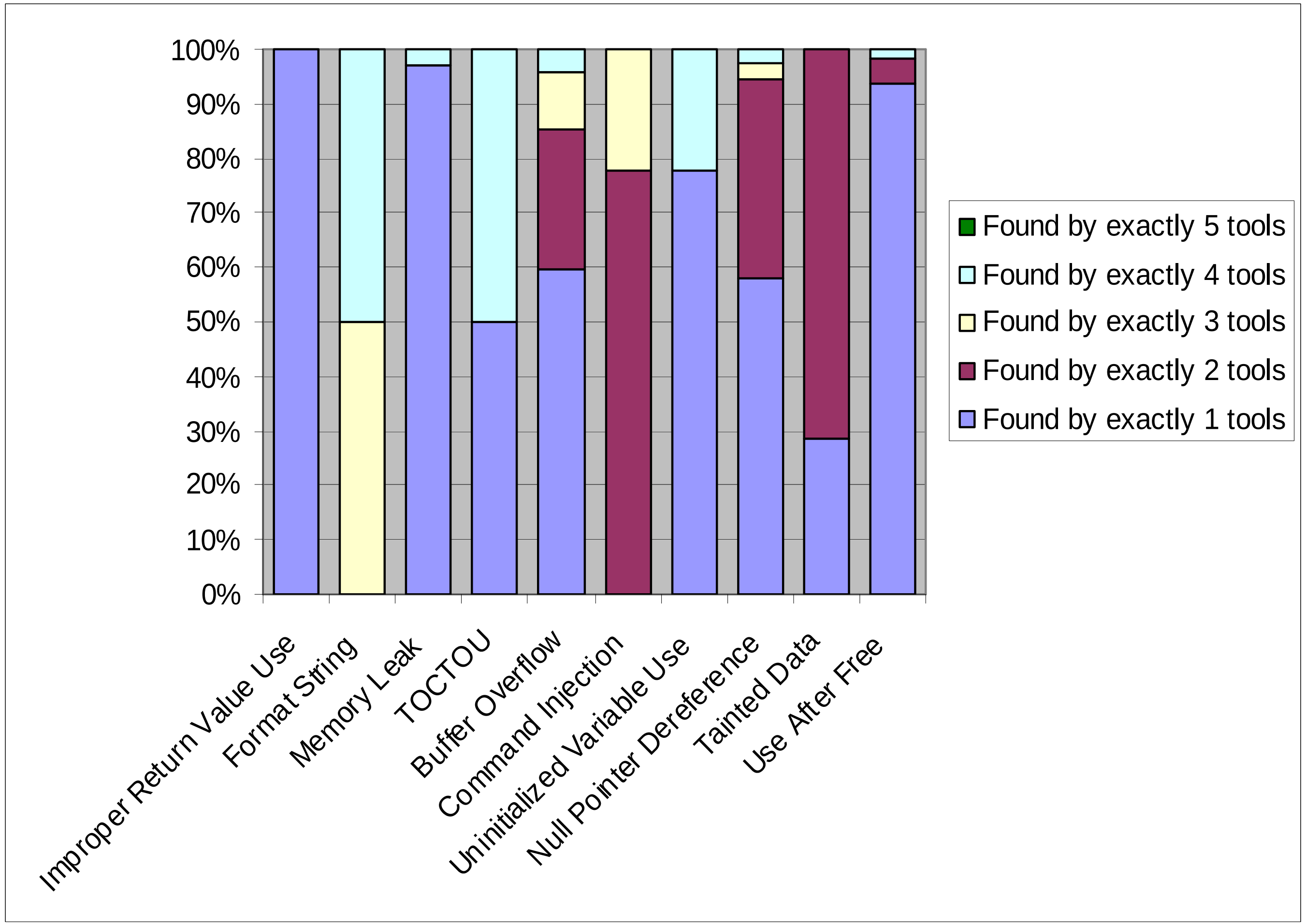
Reliability

- Heuristically solvable but difficult problems:
  - Binary analysis
  - Inter-procedural logic
  - Multi-lingual analysis
  - Analysis in the face of loops and recursion
  - Understanding data structures
  - Understanding the run-time environment

Transparency

- Evidence that a flaw is genuine
  - Tool confidence
  - Formal proofs of correctness
  - Complete description of how and why the flaw was flagged
- Evidence that non-flagged code can be trusted
- Bounds of uncertainty
- Knowledge of what the tool doesn't detect

Flaw Detection



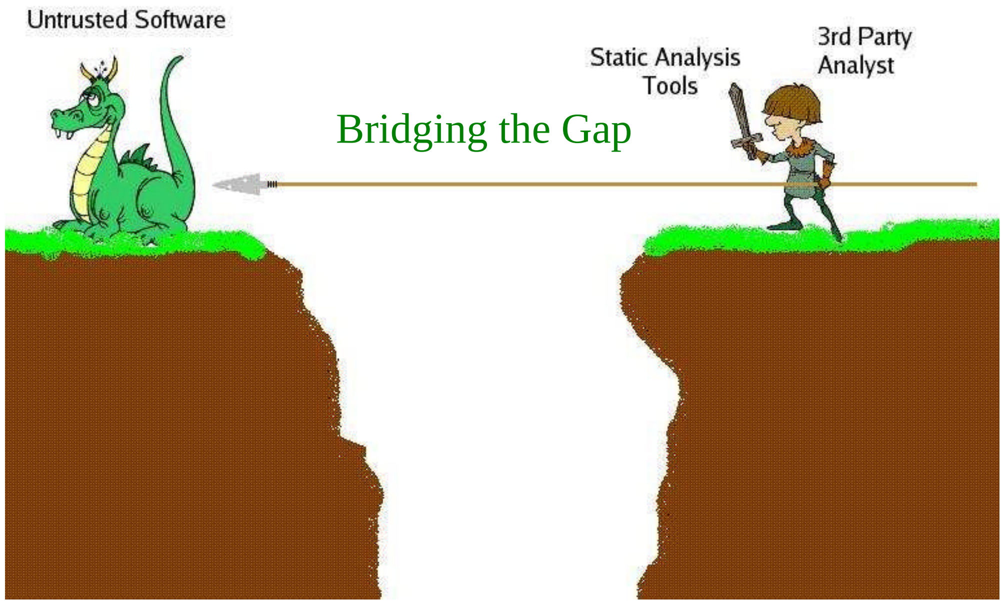
Providing World-Class Services for World-Class Competitiveness

Interoperability

- Unparsable output
- No standard for result correlation
- No standard API for:
  - New checkers
  - Data exchange
  - Review of results
  - Accessing program models

A Way Forward

- Minimum Requirements for Review Assistance
- Potential Incident Identification
- Standard Result Format
- Standard IR API
- Annotation & Assertion Support
  - Source code and tool output

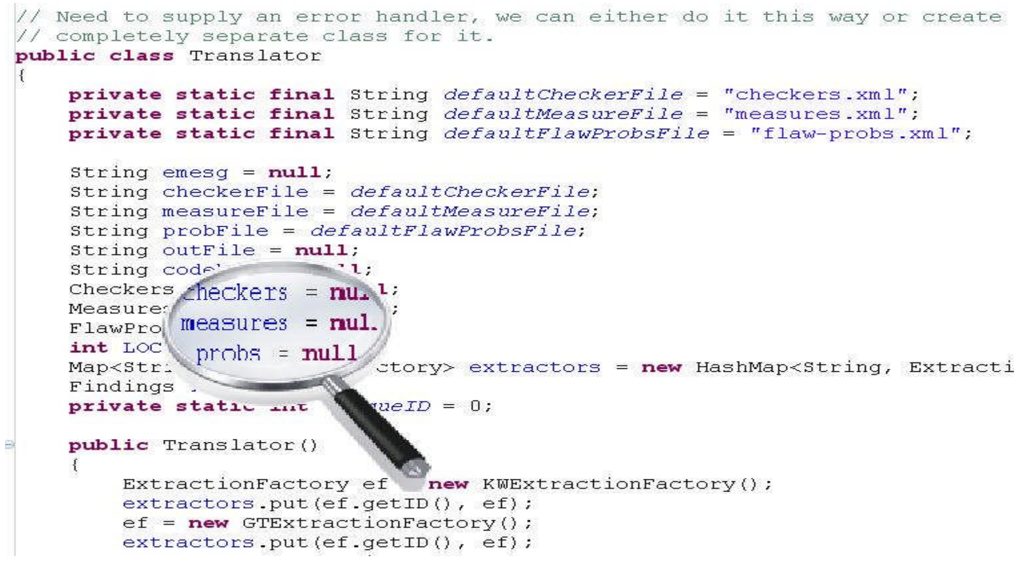


Minimum Requirements for Review Assistance

- Cross referenced browsable results with source code
- Providing pre/post conditions
- Providing flow analysis
- Supporting value sets for variables
- Supporting programming slicing
- Helping to visualize code
- Helping to easily query code
- Presenting the body of evidence of a flaw

Concurrent Technologies Corporation

Potential Incident Identification

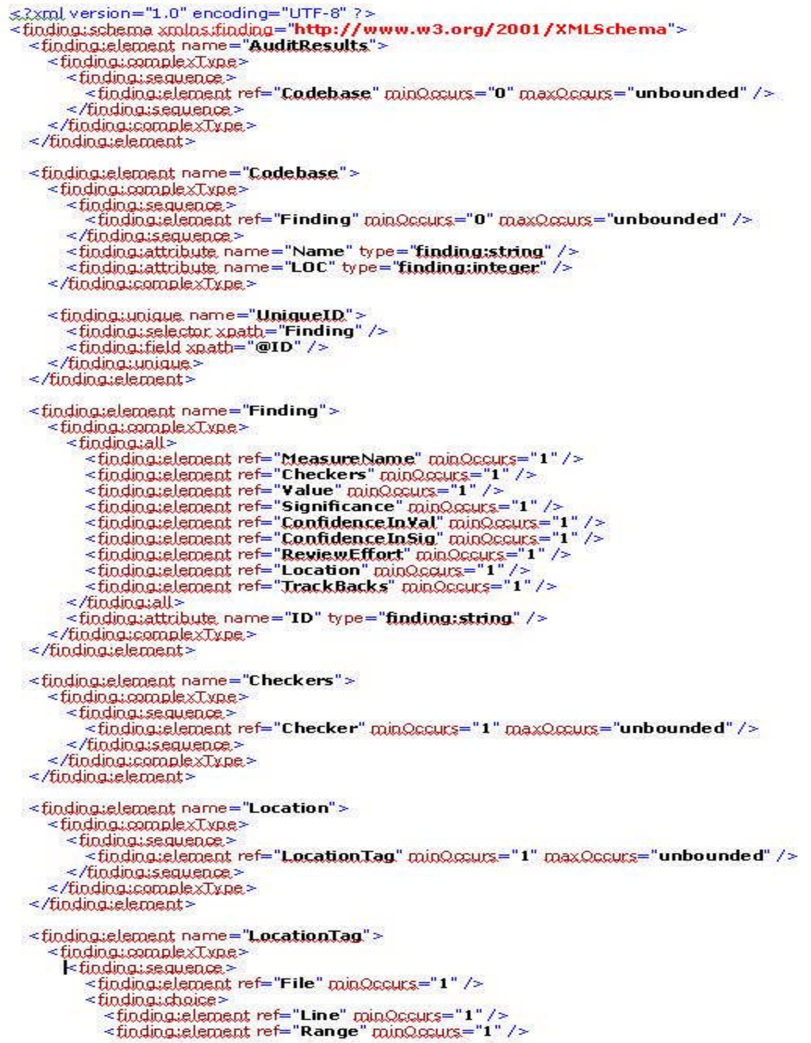


```
// Need to supply an error handler, we can either do it this way or create
// completely separate class for it.
public class Translator
{
    private static final String defaultCheckerFile = "checkers.xml";
    private static final String defaultMeasureFile = "measures.xml";
    private static final String defaultFlawProbsFile = "flaw-probs.xml";

    String emesg = null;
    String checkerFile = defaultCheckerFile;
    String measureFile = defaultMeasureFile;
    String probFile = defaultFlawProbsFile;
    String outFile = null;
    String code        ll;
    Checkers checkers = null;
    Measures measures = null;
    FlawPro  probs = null
    int LOC
    Map<Stri        ctory> extractors = new HashMap<String, Extracti
    Findings
    private static int        ueID = 0;

    public Translator()
    {
        ExtractionFactory ef    new KWExtractionFactory();
        extractors.put(ef.getID(), ef);
        ef = new GTExtractionFactory();
        extractors.put(ef.getID(), ef);
```

- Identify all constructs that could cause a particular type of flaw
- Helps to determine how often the programmers “got it right”
- Helps in estimating how much the analysis covered and the breadth of exposure

Standard Result Format

- Addresses:
  - Interoperability
  - Review assistance
- Requires:
  - Std. source locations
  - Std. line counting
  - Std. flaw types
  - Std. characterizations
- Provides:
  - Data exchange
  - Collaboration
  - External audit



```xml
<?xml version="1.0" encoding="UTF-8" ?>
<finding:schema xmlns:finding="http://www.w3.org/2001/XMLSchema">
  <finding:element name="AuditResults">
    <finding:complexType>
      <finding:sequence>
        <finding:element ref="Codebase" minOccurs="0" maxOccurs="unbounded" />
      </finding:sequence>
    </finding:complexType>
  </finding:element>

  <finding:element name="Codebase">
    <finding:complexType>
      <finding:sequence>
        <finding:element ref="Finding" minOccurs="0" maxOccurs="unbounded" />
      </finding:sequence>
      <finding:attribute name="Name" type="finding:string" />
      <finding:attribute name="LOC" type="finding:integer" />
    </finding:complexType>

    <finding:unique name="UniqueID">
      <finding:selector xpath="Finding" />
      <finding:field xpath="@ID" />
    </finding:unique>
  </finding:element>

  <finding:element name="Finding">
    <finding:complexType>
      <finding:all>
        <finding:element ref="MeasureName" minOccurs="1" />
        <finding:element ref="Checkers" minOccurs="1" />
        <finding:element ref="Value" minOccurs="1" />
        <finding:element ref="Significance" minOccurs="1" />
        <finding:element ref="ConfidenceInVal" minOccurs="1" />
        <finding:element ref="ConfidenceInSig" minOccurs="1" />
        <finding:element ref="ReviewEffort" minOccurs="1" />
        <finding:element ref="Location" minOccurs="1" />
        <finding:element ref="TrackBacks" minOccurs="1" />
      </finding:all>
      <finding:attribute name="ID" type="finding:string" />
    </finding:complexType>
  </finding:element>

  <finding:element name="Checkers">
    <finding:complexType>
      <finding:sequence>
        <finding:element ref="Checker" minOccurs="1" maxOccurs="unbounded" />
      </finding:sequence>
    </finding:complexType>
  </finding:element>

  <finding:element name="Location">
    <finding:complexType>
      <finding:sequence>
        <finding:element ref="LocationTag" minOccurs="1" maxOccurs="unbounded" />
      </finding:sequence>
    </finding:complexType>
  </finding:element>

  <finding:element name="LocationTag">
    <finding:complexType>
      <finding:sequence>
        <finding:element ref="File" minOccurs="1" />
        <finding:choice>
          <finding:element ref="Line" minOccurs="1" />
          <finding:element ref="Range" minOccurs="1" />
```

Standard IR API

- Addresses:
  - Interoperability
  - Extensibility
- Requires:
  - Vendor and Community buy in
  - Funding?
- Provides:
  - Customization
  - Technology Transfer
  - Speeds tool development

Annotation Support

- Tool support for language annotations
  - Verifying assertions
  - Inferring design intent
- Tool support for generating annotations
  - Verifiable assertions about the code
  - Provide annotations to assist code review

Summary

- Current state of software assurance demands much but there are limited practical solutions
- Static analysis tools offer a promise to help but must first overcome issues of reliability, transparency, detection and interoperability
- Common open data formats and APIs can enhance the tools and improve our ability to use them