



*Concurrent  
Technologies  
Corporation*

# **Gaps in Static Analysis Tools**

**Dr. Christopher Telfer**



1

*Providing World-Class Services for  
World-Class Competitiveness*

# Overview

- **Perspective**
- **Desirable Characteristics**
- **Significant Capability Gaps**
- **A Way Forward**
- **Summary**

# Perspective: Experience with the Tools

- Collaborative effort with Carnegie Mellon CyLab
- Thoroughly examined and compared five (5) commercial static analysis tools
  - Subjected to a variety of tests involving both synthetic and “real-world” code
- Participating in the development of a methodology for extracting assurance about software using static analysis



# Perspective: Tool Usage Model

- 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 cut across all tools:
  - Low False Positive and False Negative rates
  - Deep coverage of at least one flaw type
  - Wider coverage of major security flaw types
  - Ease of use
  - Support for major programming languages
  - Tunable



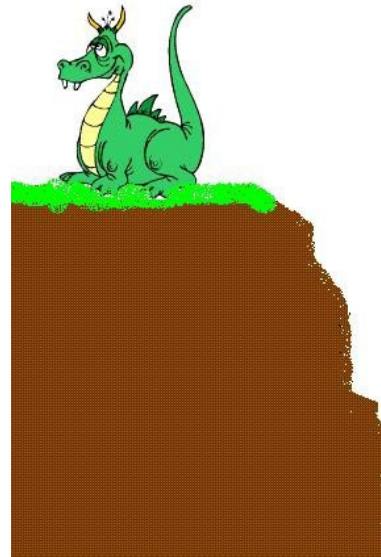
# Existing Features (that we like)

- Code understanding facilities
- Extensibility
- Interprocedural analysis
- Review/Audit Support
- Modeling
- Flow analysis
- Flaw pre/post conditions
- Data export & import
- Code browsing
- Security knowledge-base
- 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 the tool detect flaws of a certain type?
- Interoperability
  - Can we get multiple tools to work together?

Untrusted Software



Static Analysis Tools  
3rd Party Analyst



# Reliability

- Really Hard problems:
  - Handling in-line assembly
  - Pointer arithmetic
  - Code generated or loaded at run time
  - Context sensitive DF/AR/VS analysis

```
void doit() {  
    int true = 1;  
    asm("mov $0, -4(%ebp)");  
    if ( true ) .....
```





# Reliability

- Heuristically solvable but difficult problems:
  - Programs composed of multiple languages
  - Inter-procedural logic
  - Loop and recursion analysis
  - Understanding data structures
  - Understanding the run-time environment
    - Systems security
    - Hardware, OS, DLLs, Collaborating Apps ....

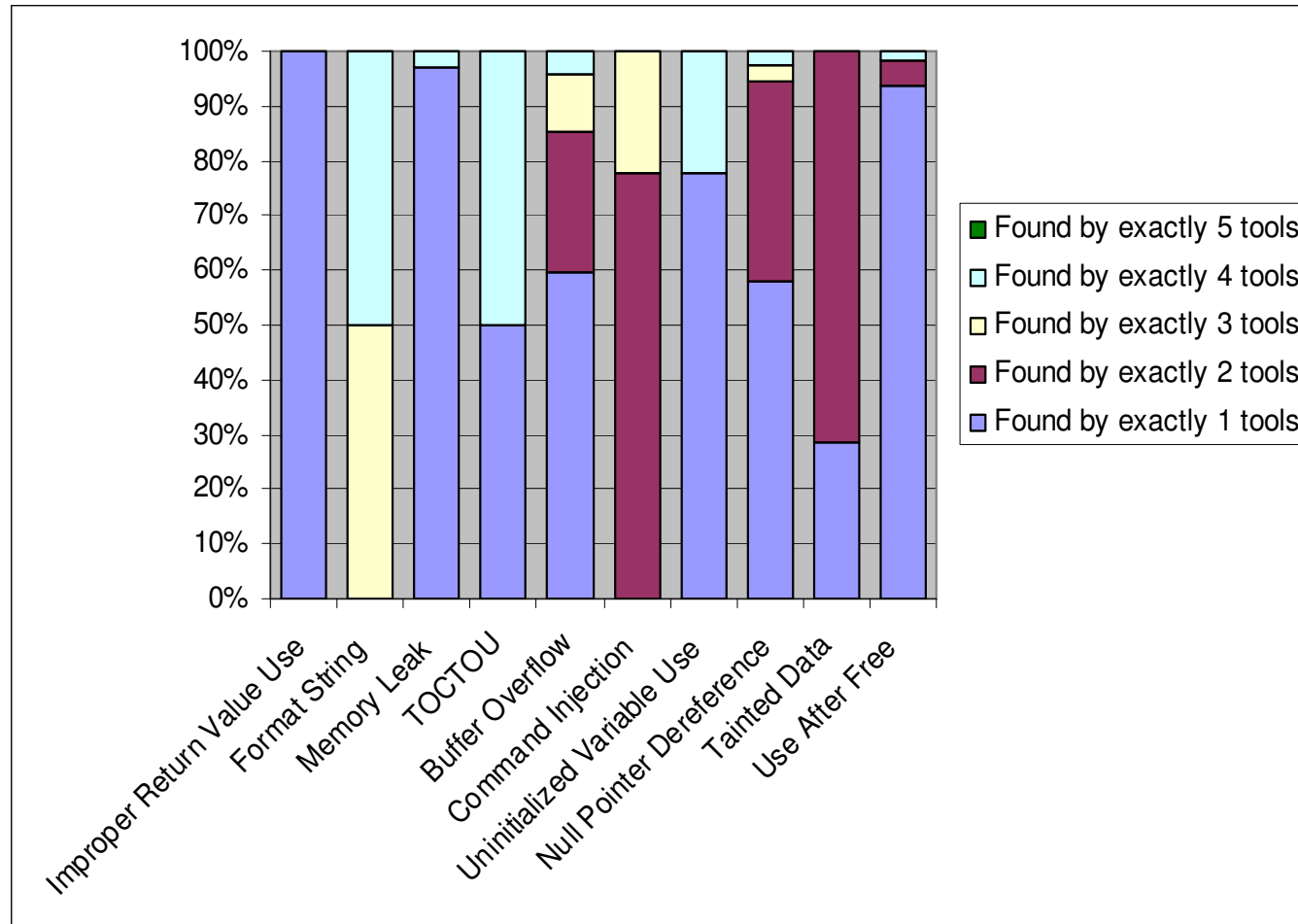


# Transparency

- Evidence that a flaw is genuine
  - Tool confidence
  - Formal proofs of correctness
  - Complete description of how and why the flaw was flagged
- Evidence that a flaw does not exist
  - At some place and context in the code
- Knowledge of what the tool can't or doesn't detect
  - Knowledge of where tool bails out
  - Limits of analysis



# Flaw Detection



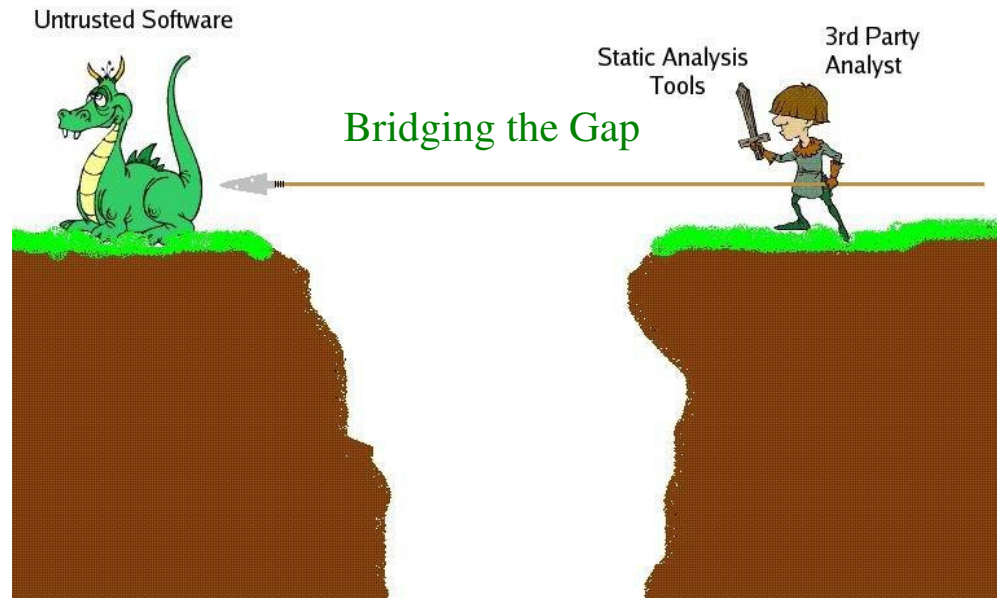
# Interoperability

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



# A Way Forward

- Minimum requirements for review assistance
- Potential incident identification
- Standard result format
- Standard IR API
- Annotation & assertion support



# Minimum Requirements for Review Assistance

- Navigation through code
- Presenting the body of evidence of a flaw
- Providing control/data flow analysis
- Providing pre/post conditions and assertions
- Helping to visualize code
- Helping to easily query code

# 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 = null;
    Checkers checkers = null;
    Measures measures = null;
    FlawProbs probs = null;
    Map<String, ExtractionFactory> extractors = new HashMap<String, ExtractionFactory>();
    Findings findings = null;
    private static int sequenceID = 0;

    public Translator()
    {
        ExtractionFactory ef = new KWExtractionFactory();
        extractors.put(ef.getID(), ef);
        ef = new GTEExtractionFactory();
        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 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:complexType>
  </finding:element>

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

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



# Standard IR API

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



# Annotation Support

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



# Ongoing Initiatives

- SwA Ecosystem
- CWE
- SAMATE
- SwA Consumer Consortium
- Others



# Summary

- Current state of software assurance demands much but provides 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

