Tool Output Integration Framework (TOIF)

Data Access Technologies
Ed Seidewitz
KDM Analytics
Dr. Nikolai Mansourov
• Problem
  – Effective and systematic measurement of the risks posed by software vulnerabilities

• Challenge
  – One of the key challenges is that analysis solution consists of multiple tools, information sources and services that are currently fragmented lacking intuitive and efficient integration due to
    • Inconsistency in the nomenclature of reported vulnerabilities caused by ambiguity of vulnerability definitions – inconsistency in interpretation of Common Weakness Enumeration (CWE) instances
    • Lack of agreement on what are the parts of vulnerability to report – what constitutes vulnerability report
    • Lack of interoperability that is based on common definition of system artifacts
Moving toward Solution: The Tools’ Output Integration Framework (TOIF):

• Creating bases for composite vulnerability analysis tools on top of existing off-the-shelf vulnerability detection tools
  – E.g. Applied Visions
• Improving the breadth and accuracy of vulnerability analysis
• Improving the rigor of assessments by bringing vulnerability detection into architecture context
• Normalizing vulnerability reporting protocols
• Leveraging OMG Software Assurance Ecosystem standards, Software Fault Patterns (SFPs) and CWEs

Delivering
• open source product:
  • analyzer and run time framework for integrating findings of vulnerability detection tools
  • integration of 5 existing open source tools
• framework integration of proprietary tools in area of architecture and risk analysis to show greater value when viewing CWEs within the architecture and risk content
The Tools’ Output Integration Framework

Fact-oriented integration

- Capability to integrate multiple vulnerability detection tools as “data feeds” into the repository
  - Based on a common protocol for exchanging vulnerability findings
  - Achieved through normalizing vocabularies across multiple tools
- Capability to collate findings from several tools
- Capability to put vulnerability findings into the context of other facts about the system (such as metrics, architecture, design patterns, etc.)
  - Based on existing standard protocol for exchanging system facts, the OMG Knowledge Discovery Metamodel (KDM),
  - now ISO/IEC 19506
- As the result: single integrated repository of high-fidelity facts about a software system
Semantic integration focuses on facts

Vulnerability detection tools

- CPPcheck
- FindBugs
- JLint
- RATS
- Splint

TOIF adapters (normalization) → TOIF XMI

Build environment → Code

• Finding
• Code Location
• File
• Line Number
• Name
• Directory
• CWE id
• SFP id
• Fault Cluster id
• Statement
• Data Element

• Finding references Code Location
• Finding has CWE id
• Code Location references File
• Code Location references Line Number
• File has Name
• Finding involves Statement
Example: Wireshark

- **Statistics**
- Wireshark ~ 2MLOC
- Total files analyzed: 1519
- Run 3 open source tools: cppcheck, splint and RATS- number of findings: 18949
  - Cppcheck reported 7051 issues
  - Splint reported 10917 issues
  - RATS reported 981 issues

- How to make sense out of it?
  - Identify overlaps and unique findings
  - Focus on the findings that matter
  - Prioritizing findings
Example of findings

Source:
1150 fputs("%% the page title\n", output->fh);
1151 ps_clean_string(psbuffer, filename, MAX_PS_LINE_LENGTH);
1152 fprintf(output->fh, "/ws pagetitle (%s – Wireshark \\
    VERSION "%s") def \n",
    psbuffer, wireshark_svnversion);
1153 fputs("\n", output->fh);

RATS report on line # 1152:
“Check to make sure that the non-constant format string passed as argument 2 to this
function call does not come from an untrusted source."

SFP-24; CWE-134

SPLINT report on line # 1152:
"Format argument 1 to fprintf (%s) expects char * gets
unsigned char [256]: psbuffer"

SFP-1; CWE-681

Same line number, different weakness

SFP-24; CWE-134

SFP-1; CWE-681
```c
print_ps_preamble(output->fh);

fputc("% the page title\n", output->fh);
ps_clean_string(pbuffer, filename, MAX_PS_LINE_LENGTH);

fprintf(output->fh, "%s pagetitle (%s - Wireshark " VERSION "%s" def\n", ps
fputs("\n", output->fh);
return !ferror(output->fh);

static gboolean
print_line_ps(print_stream_t *self, int indent, const char *line)
{
    output_ps *output = (output_ps *)self->data;

    unsigned char pbuffer[MAX_PS_LINE_LENGTH]; /* static sized buffer */

    ps_clean_string(pbuffer, line, MAX_PS_LINE_LENGTH);
    fprintf(output->fh, "%d (%s) putline\n", indent, pbuffer);

```
Findings in the Context of Architecture

Consider the following:
- architecture component where issue is identified
- distance from injury (based on SFP description)
- distance from entry point into system
Potential Benefits

• Integration of existing vulnerability detection tools and cross-correlation of their findings with architectural risk analysis is important for software assurance

• Powerful open source vulnerability detection platform

• Reference implementation for standard-based adaptors
  – Blue print how to integrate additional analyzers

• Adoption of standard-based reporting of vulnerabilities

• Utilization of open source development to advance the SwA space
• Commercialization through open source
  – Integrate selected open source vulnerability detection tools
  – Open source KDM extraction tools
• TOI Framework protocol (TOIF XMI) is easy to adopt by tool vendors
• Deliverables:
  – a ready-to-use open source composite vulnerability analyzer integrating 5 existing open source vulnerability detection tools
  – integrating proprietary architecture analysis tool
  – a protocol for exchanging vulnerability findings
  – blueprints for adaptors of the protocol
  – practical usability and accuracy data based on the case study