Truth is Subjective

SATE 2009 Experience with CodeSonar®

Presented by:
Paul Anderson
Outline

- Introduction to CodeSonar
- Warning correctness
  - Examples of warnings
- Suggestions for the future
CodeSonar

- Advanced Static Analysis for C/C++
- Oriented towards general-purpose bug finding
  - Particularly for embedded/safety-critical
- Not specialized for finding software security issues
  - Although there is much overlap
    - Buffer overruns, Null pointer dereferences, Uninitialized variable, Race conditions, etc.

- Analysis techniques
  - Whole program model
  - Symbolic execution
    - Flow-, Context-, and Path-sensitive

- Designed for high scalability and low false positives
  - At the sacrifice of soundness

- Highly configurable and customizable
Warning “correctness” judgment

- Study used *true, false, insignificant*
- Judgment categories strongly depend on role of the analyst:
  - Code author
  - Code reviewer
  - QA dept
  - Internal security reviewer
  - External security analyst
  - Attacker

- Nature of application affects judgments too:
  - Safety-critical
  - Real-time
  - High security
CodeSonar Warning

- Buffer overrun reported in Irssi:
  - struct tm tm;
  - memcpy(&tm, localtime(&now), sizeof(tm));

- No buffer overrun possible

- Caused by operator error!
  - Mismatch between sizes of types
  - Model for localtime based on 32-bit pointers, but analysis done in a 64-bit environment
  - Once corrected, this and several other warnings not reported
CodeSonar misjudged warning

```
if (rl + l >= rm) {
    rm = rl + l + 1;
    r = TREALLOC(r, rm, char);
}
strncpy(r + rl, vv, l);     /* Null Pointer Dereference */
```

Judged by evaluators as false positive.

But the TREALLOC may return NULL, so a NPD is possible.
int pvm_pkstr(cp)
    char *cp;
{
    int l = strlen(cp) + 1;
    int cc;

    Buffer Underrun reported

    p = getenv("PVM_EXPORT");
    ...
    p = p - 11;
    ...
    pvm_pkstr(p);

    Suspicious code!

    SATE reporting format obscured the real reason:

    This code does work…

    p = “xyz”
    p - 11 = “PVM_EXPORT=xyz”
Getenv() issue

char environ[] = “USER=paul\0PVM_EXPORT=xyz\0PATH=/usr/bin... 

Possible judgments:
- False positive because the target platform works this way?
- True positive because this may not port?
- Insignificant?
SATE reviewer judgments

- Out of 23 false judgments in one benchmark, 11 are disputed
- Insignificant and true judgments not reviewed

Recommendation:
- Future Expos judge results from multiple perspectives
Suggestions for the future

- Keep SATE as it is
  - Great for vendors
    - I get to brag about CodeSonar
- Run a high-profile competition
  - Big cash prizes => lots of publicity => raises awareness
- Extend SAMATE Reference Dataset (SRD)
  - Potential to have very wide benefit to all vendors
  - Potential to spur research into new techniques
Extend SRD

- Need real examples of bugs that matter
  - Boiled down or abstracted examples are much less useful
  - As are samples with injected flaws
- Dataset would be useful for other approaches
  - Including some we haven’t thought of yet
- Base the Expo around these samples
Ideal Specimen

- A serious bug that was observed in the wild
  - With cross reference to CVE
- Full source code and build system for the vulnerable program
  - plus full source code for dependences
  - and a description of the platform and toolchain used to build
- A full explanation of the bug
  - Referencing locations in the source
  - Relevant CWE entry
  - History of how it was found
- A patch that fixes the bug, and **only** that bug
- An executable in which the bug was observed in the wild
  - plus one in which it was fixed
The End
SATE Stated Goals

- **Goals**
  - To enable empirical research based on large test sets
  - To encourage improvement of tools
  - To speed adoption of tools by objectively demonstrating their use on real software

- **Our goal is not to evaluate nor choose the "best" tools.**

- **Characteristics to be considered**
  - Relevance of warnings to security
  - Correctness of warnings
  - Prioritization of warnings
Customer Evaluation Methodology

- Does the tool integrate with my build system?
  - Can it identify all the code that is compiled?
  - Does it model the compiler properly?
- Does it find interesting bugs?
- Is precision and recall acceptable?
- Does it make triage easy?
  - Evidence for conclusion
  - UI for understanding warnings and related code
- Can I add new checks?
- Can managers track progress?
- Does it integrate with my bug-tracking system?
- Is the ROI appropriate?
Customer Use Methodology

- Run the analysis tool on the code
- Eyeball the results, and assess
  - Is there code that should be incorporated?
    - E.g., Irssi uses glib
    - Either add the code, or model it
  - Are there classes that are uninteresting?
    - E.g., unsafe casts in Irssi
    - Set up filters; adjust default priorities
  - Are there parameters to adjust?
    - E.g., may malloc() return NULL?
  - Are there custom checks?
  - Is the workflow optimal?
- Iterate until satisfied
- Put tool into production