A Test Suite for Basic CWE Effectiveness

Paul E. Black

paul.black@nist.gov

http://samate.nist.gov/

Static Analysis Tool Exposition (SATE V) News

- **We choose test cases by end of May**
- **Tool output uploaded to NIST web site by end of July (end of August at latest)**
- **Use Coverage Claims Representation**
- **Experience workshop in early 2014**
- **Prizes for best test cases**
- ***We invite***
  - ***tool users to be on planning committee***
  - ***static analysis tool makers to participate***

Statistics, the Universe of Programs, and Everything Relevant (SUPER)

- **We want a set of small programs that statistically represent all programs.**
- **But, what is the universe, i.e., "all programs"?**
- **All strings of characters? All programs that compile? Everything a competent programmer would write? Including malware and obfuscated code? Generated code?**
- **What statistics do we want? Can we even pose reasonable questions?**
- **Let's get statisticians, computer scientists, and others together in a workshop.**

Common Weakness Enumeration (CWE)

- **A "dictionary" of every kind of bug or flaw in software.**
- **More than 600 distinct classes**
  - **For instance, buffer overflow, OS injection, race condition, cross-site scripting, directory traversal, left-over debug code, hard-coded password, and insecure random number generator.**

http://cwe.mitre.org/

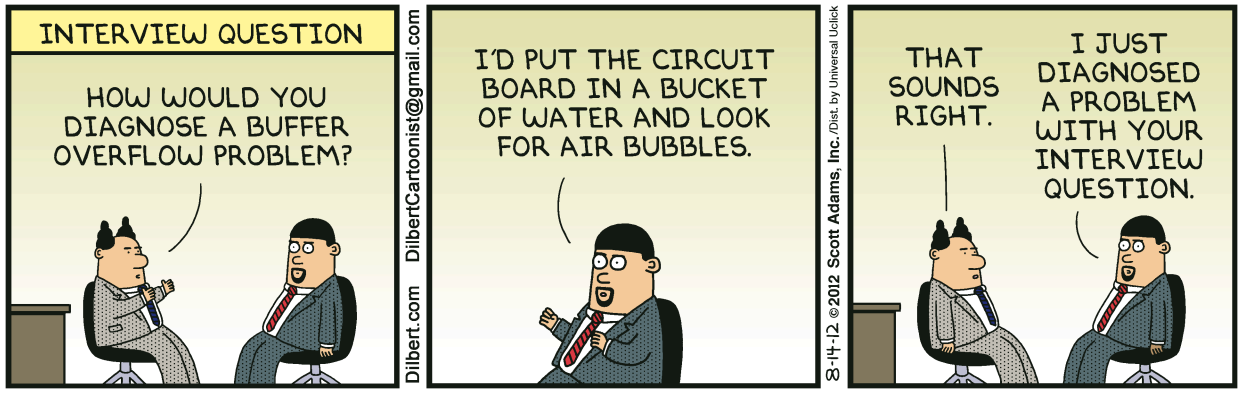
MITRE's CWE Compatibility and Effectiveness Program

- **Phase 1 – Declare compatibility**
- **Phase 2 – Verify mapping to CWEs**
- **Phase 3 – Test cases whose results "allow [one] to understand which CWE identifiers [a] capability is effective in locating"**

http://cwe.mitre.org/compatible/program.html

Start With CWE-121 Stack-based Buffer Overflow in C Language

- **It is a frequent, serious problem.**
- **It is well-defined and easily understood.**
- **We have thousands of examples.**
- **It is addressed by static analysis, compile-time techniques, or run-time detection.**



We Propose a Test Suite for Basic CWE Effectiveness

What is the Test Suite Like?

- **Publicly available**
- **Consists of a small number of programs**
- **Each program is**
  - **short**
  - **code is vulnerable, i.e., exploitable**
  - **standard code (no language extensions)**
  - **(usually) synthetic**
  - **fairly "clean"**

Background Work

- **Over the summer NIST researchers installed five static analyzers, then examined 7,338 cases in 9,962 files from**
  - **Juliet (split into 5,892 good & bad cases)**
  - **Kratkiewicz (1,139 cases)**
  - **KDMA TCG (249 cases)**
  - **2005 Fortify (41 cases)**
  - **other SRD (17 cases)**

Proposed CWE-121 Basic Set

- **It consists of five cases.**

- **The most basic case is basic-00001-min.c**

```
char buf[10];
buf[10] = 'A';
```

Two More Cases

- **basic-00034-min.c**
  - **access through a pointer**

```
*(buf + 10) = 'A';
```

- **basic-00045-min.c**
  - **library function**

```
strcpy(buf, "AAAAAAAAAA");
```

Last Two Cases

- **basic-00182-min.c**
  - **fgets(): limited copy and external input**

```
fgets(buf, 11, f);
```

- **stack_overflow_loop.c**
  - **variable index; bad bounds check**

```
for (unsigned i=1;i<=10;++i) {
    bStr[i] = (char)i + 'A';
```

Next Step – Code Complexities

- **to "articulate what types of complexity in software [a] capability is most successful at dealing with ..."**

What is “Code Complexity”?

```
char data;

data = 'C';


data = 'Z';
printHexCharLine(data);
```

```
char data;
if (1) {
  data = 'C';
} else {
  data = 'C';
  printHexCharLine(data);
}
if (1) {
  data = 'Z';
  printHexCharLine(data);
} else {
  printHexCharLine(data);
}
```

CWE-563 Unused Variable, after SRD test cases 35455 and 35456

Complexity Factors

- **Other fns: str(n)cpy/cat, memcpy/move, s(n)printf**
- **Separate files (caseA.c & caseB.c)**
- **Duplicate variable or function names**
- **Dynamic allocation (alloca); other scopes**
- **Complex expressions in index, allocation, tests, etc.**
- **Data Types**
- **Containers (buffer in struct)**
- **Unreachable and dead code**
- **Open coded or obfuscated str(n)cpy()**
- **Weakness vs. Vulnerability**

More Complexity Factors

- **Unflawed cases (for false positive)**
- **Minimal ("fake") parsing**
- **Alias (passed through other variables)**
- **Access by array or pointer**
- **Read or write access**
- **Gravity (just outside buffer or far beyond)**
- **Continuity (every element or jumping)**
- **Control flow & loop structure**
- **Intervening path**
- **Taint source**

What about tool “short cuts”?

- **Tool makers may build to a public, static set.**
  - **A secret or dynamic set has other problems.**
- **Change comments and identifier names for every download?**
- **Add innocuous statements?**
- **Transform code, like unroll loops?**

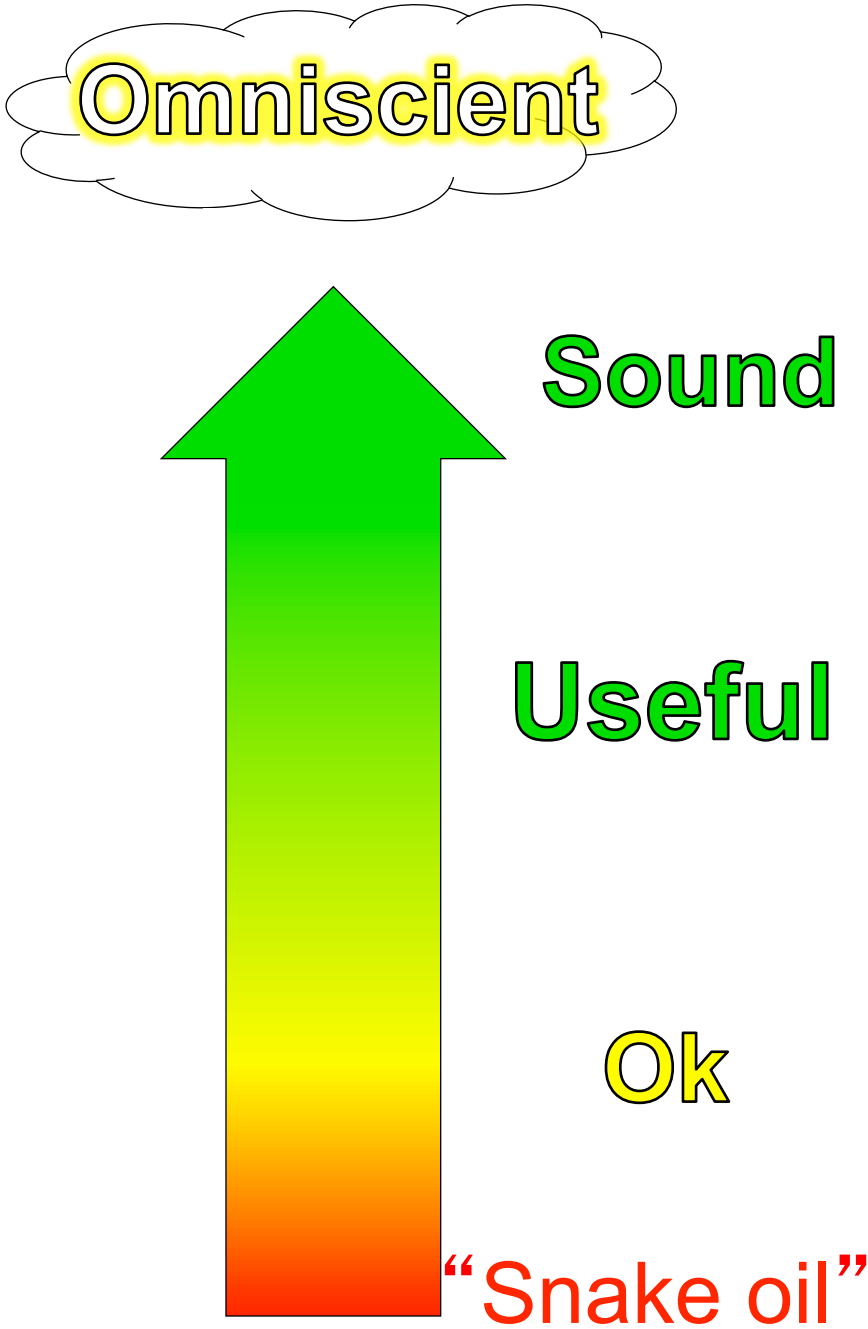
Proposal:

- **If concerns arise, privately corroborate results.**

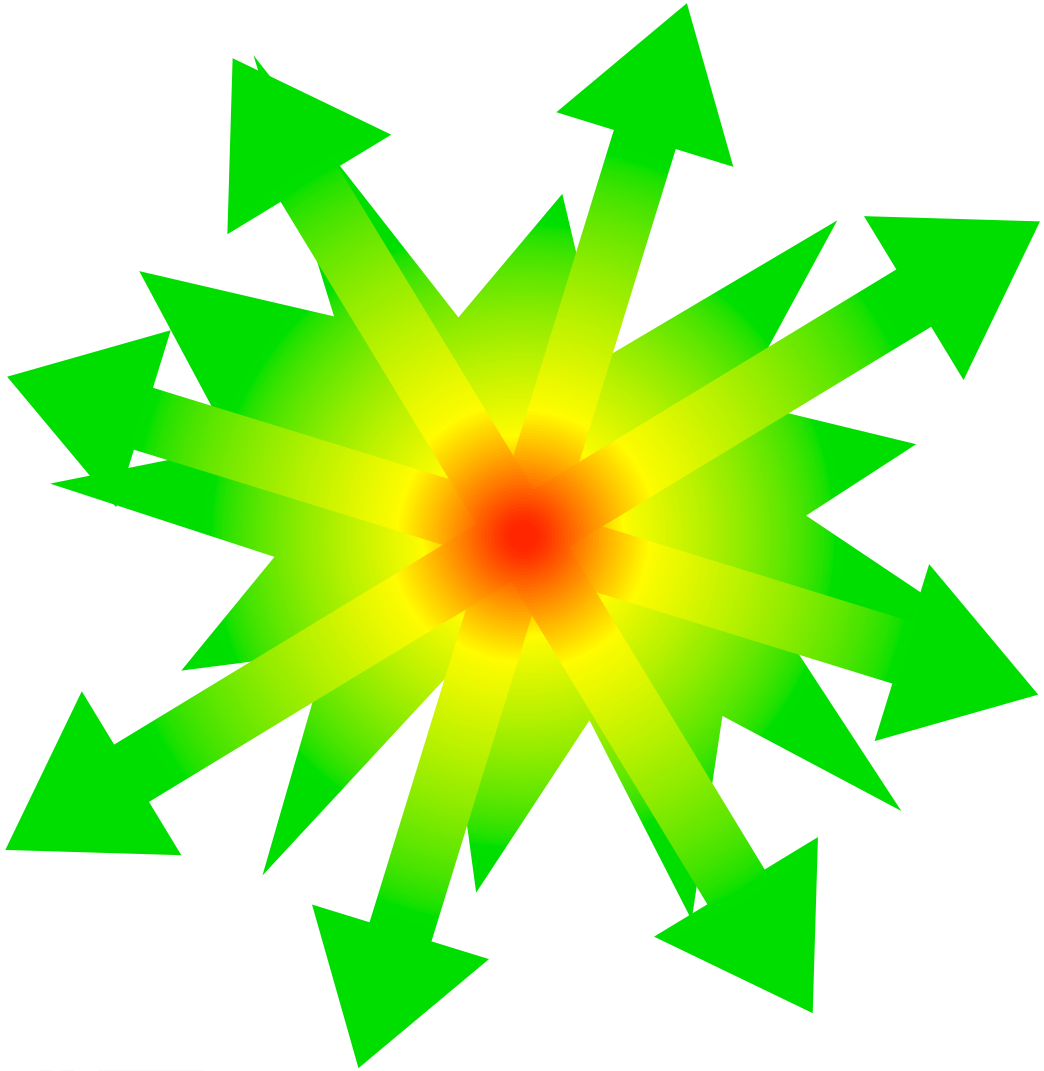
The Problem

- **Sound analysis is not perfect anyway.**
- **No test set can show all possible bugs, heuristics, variants, and corner cases.**

- ***How thorough should the test set be?***

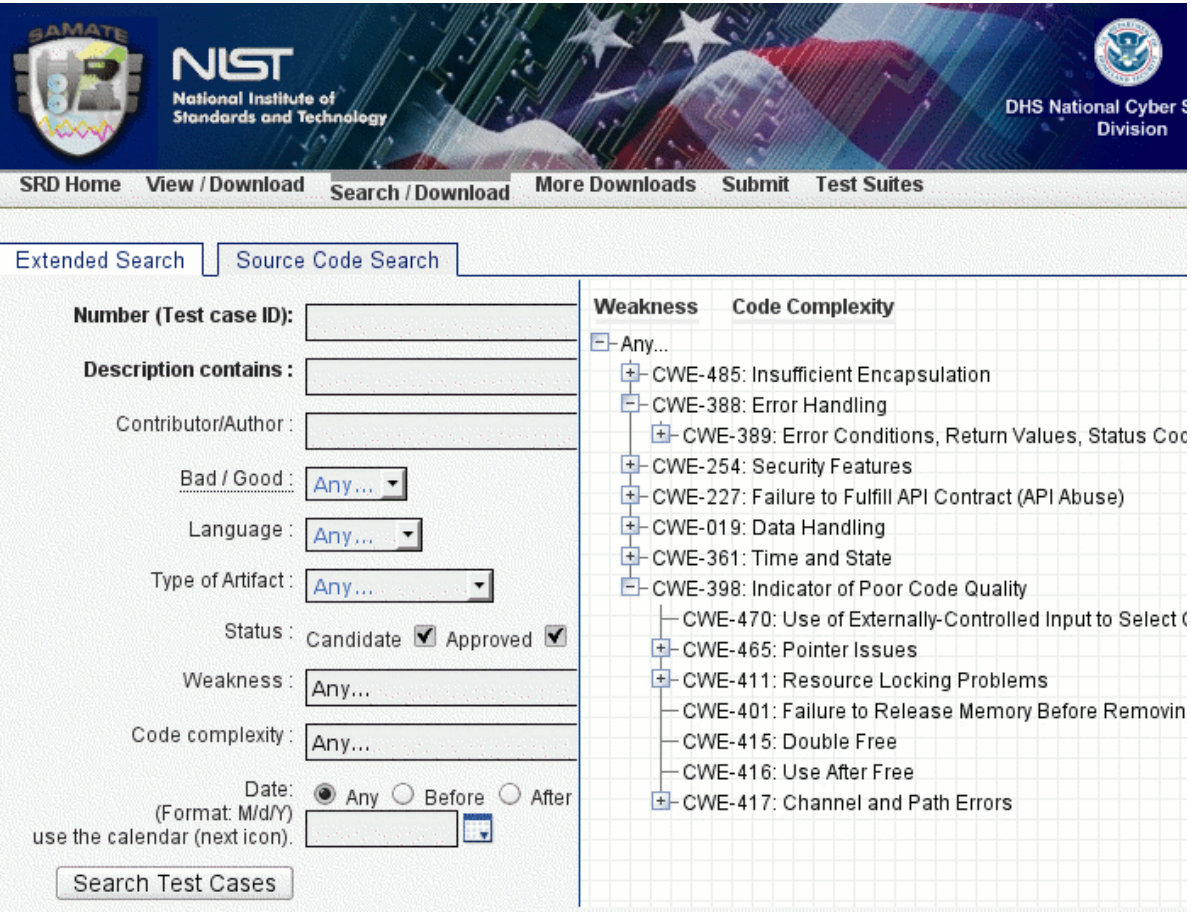


Measurement is Multidimensional



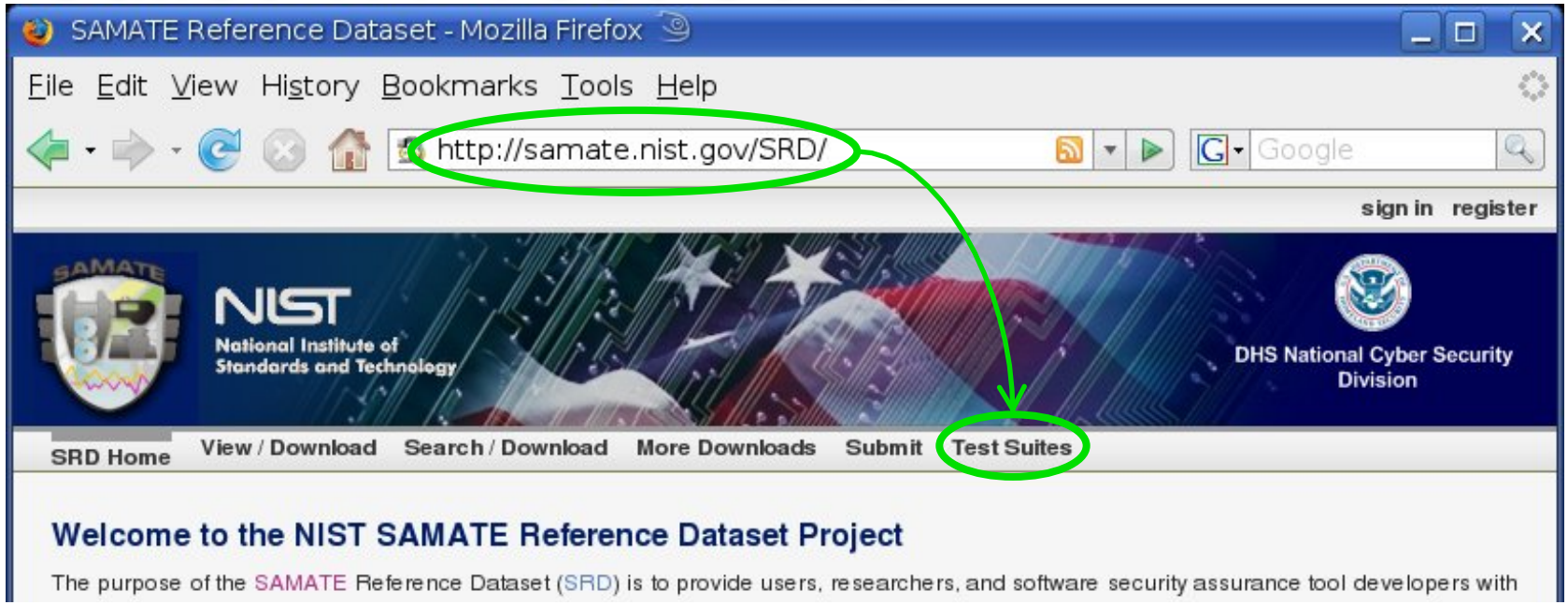
SAMATE Reference Dataset

http://samate.nist.gov/SRD/



- **Public repository for soft-assurance test cases**
- **Over 60,000 cases in C, C++, Java, C#, and Python**
- **Search and select by language, weakness, etc.**
- **Contributions from CAS, Fortify, Defence R&D Canada, Klocwork, MIT Lincoln Laboratory, Praxis, Secure Software, etc.**

Juliet 1.1 Test Suite



- **81,056 small C/C++ and Java programs covering 181 weakness classes**
- **Each case is one or two pages of code**
- **Organized by weakness, then variant, then complexity**
- **Described in IEEE Computer, Oct 2012**

STONESOUP 1 cases

- **About 460 cases in Java and C, each a program typically 200-300 lines long**
- **Cover weaknesses in Number Handling (e.g. integer overflow), Tainted Data (e.g. input validation, Injection (e.g. command injection, Buffer Overflow, and Null Pointer**
- **Each case has inputs triggering the vulnerability, as well as "safe" inputs**
- **Available as test suites**

Kratkiewicz MIT cases

- **1164 cases in C for CWE-121 Stack-Based Buffer Overflow**
- **Created to investigate static analysis and dynamic detection methods**
- **Each case is one of four variants:**
  - **access within bounds (ok)**
  - **access just outside bound (min)**
  - **somewhat outside bound (med)**
  - **far outside bound (large)**
- **Code complexities: index, type, control, …**

Other SRD Content

- **Zitser, Lippmann, & Leek MIT cases**
  - **28 slices from BIND, Sendmail, WU-FTP, etc.**
- **Fortify benchmark 112 C and Java cases**
- **Klocwork benchmark 40 C cases**
- **25 cases from Defence R&D Canada**
- **Robert Seacord, "Secure Coding in C and C++" 69 cases**
- **Comprehensive, Lightweight Application Security Process (CLASP) 25 cases**
- **329 cases from our static analyzer suite**