Code Transparency and Diagnostic Capabilities

Dr. Paul E. Black

paul.black@nist.gov

http://samate.nist.gov/

Software grows faster than the increase in precision

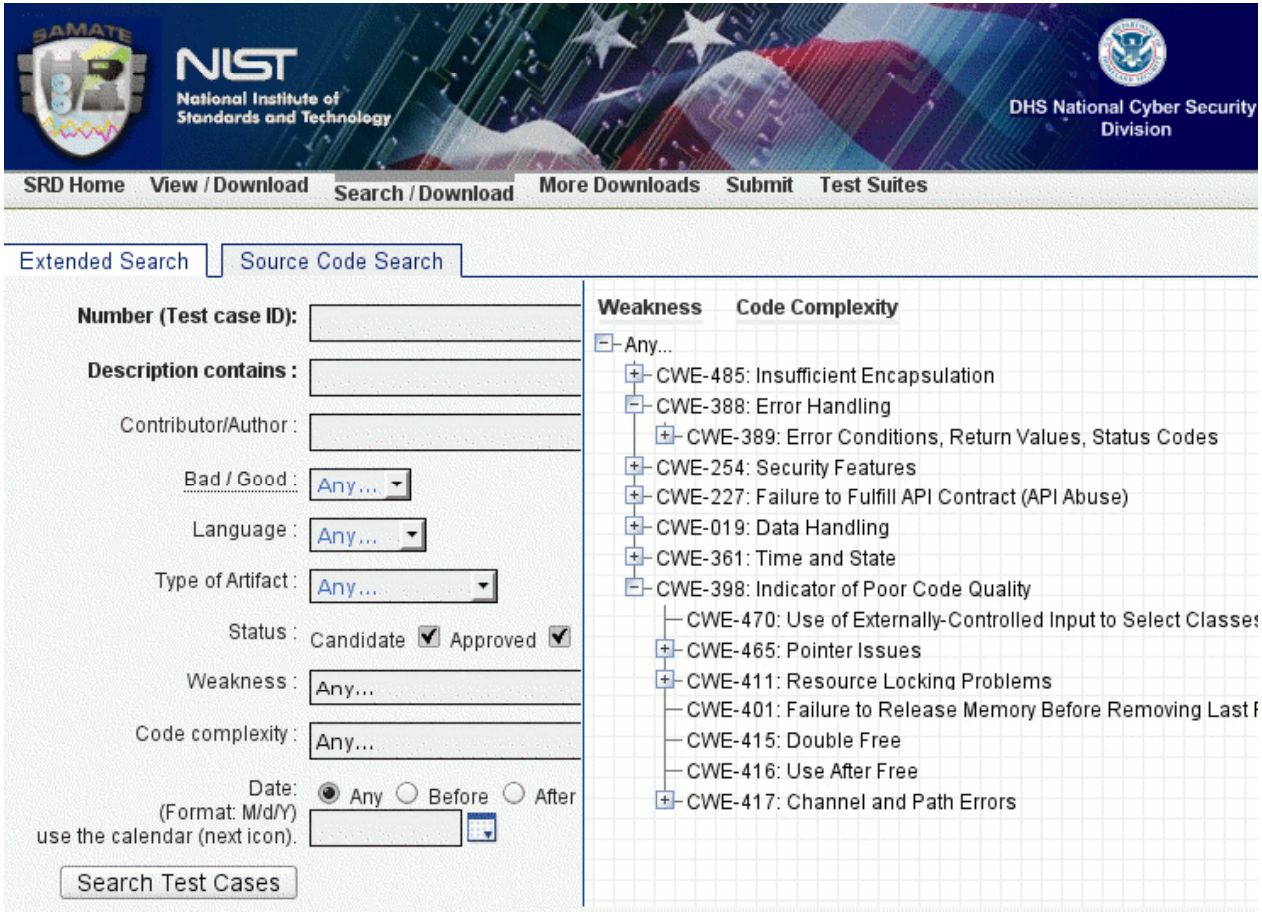
- **There are more cases to cover.**
- **Minor factors become important.**

  $\sin(x) = x - x^3/3! + x^5/5! - x^7/7! + \ldots$

- **Simplifying assumptions no longer hold.**
- **The number of interactions increases faster than number of software modules.**

  ***N* modules can have up to $N^2/2$ interactions**

Software Reference Dataset



- **Public repository for software test cases**
- **Almost 1800 cases in C, C++, Java, and Python**
- **Search and compose custom Test Suites**
- **Contributions from Fortify, Defence R&D Canada, Klocwork, MIT Lincoln Laboratory, Praxis, Secure Software, etc.**

Example: theoretical integer overflow, from case 2083

```
int main(int argc, char **argv) {
  char buf[MAXSIZE];

  . . . put a string in buf

  if (strlen(buf) + strlen(argv[2]) < MAXSIZE) {
    strcat(buf, argv[2]);
  }

    . . . do something with buf
}
```

Example: language standard vs. convention, from case 201

```
typedef struct {
  int int_field;
  char buf[10];
} my_struct;

int main(int argc, char **argv){
  my_struct s;

  s.buf[11] = 'A';

  return 0;
}
```

We need more use of tools, which are increasingly sophisticated

- **Tools for code transparency**

  … to confirm what's inside

- **Standard sets of software facts**

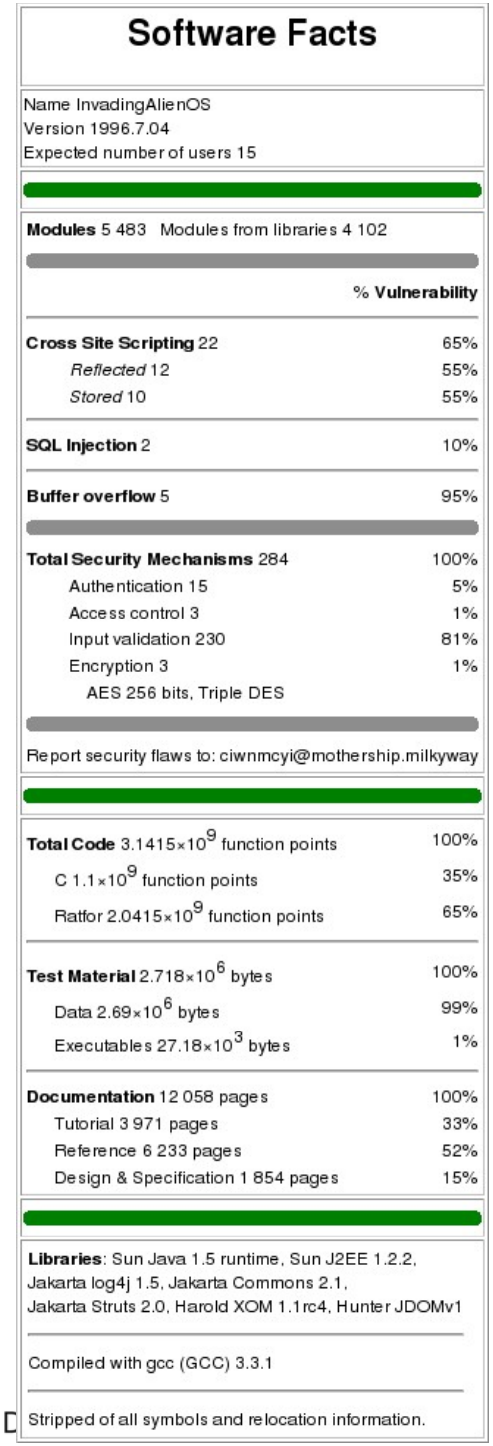
  … to communicate useful information quicker

- **Tools must be "calibrated"**
  - **NIST's Source Code Scanner specification**
  - **MITRE's Common Weakness Enumeration**
  - **NIST's Software Assurance Tool Exposition (SATE 2009)**

Software Facts

- **Software Facts should be:**
  - **Voluntary**
  - **Simple to produce**
  - **Have a standard format for other claims**
- **What could be easily supplied?**
  - **Source available? Yes/No/Escrowed**
  - **Default installation is secure?**
  - **What's accessed? (network, disk, …)**
  - **Certificates (eg, "No Severe weaknesses found by CodeChecker ver. 3.2")**
- **Cautions**
  - **A label can give false confidence.**
  - **A label may shut out better software.**
  - **Labeling diverts effort from real improvements.**
- **Hosted at Software Assurance Consortium**

http://swaconsortium.org/projects/softwareFacts/



Software Facts

Name InvadingAlienOS
Version 1996.7.04
Expected number of users 15

Modules 5 483 Modules from libraries 4 102

	% **Vulnerability**
Cross Site Scripting 22	65%
Reflected 12	55%
Stored 10	55%
SQL Injection 2	10%
Buffer overflow 5	95%
Total Security Mechanisms 284	100%
Authentication 15	5%
Access control 3	1%
Input validation 230	81%
Encryption 3	1%
AES 256 bits, Triple DES	

Report security flaws to: ciwnmcyi@mothership.milkyway

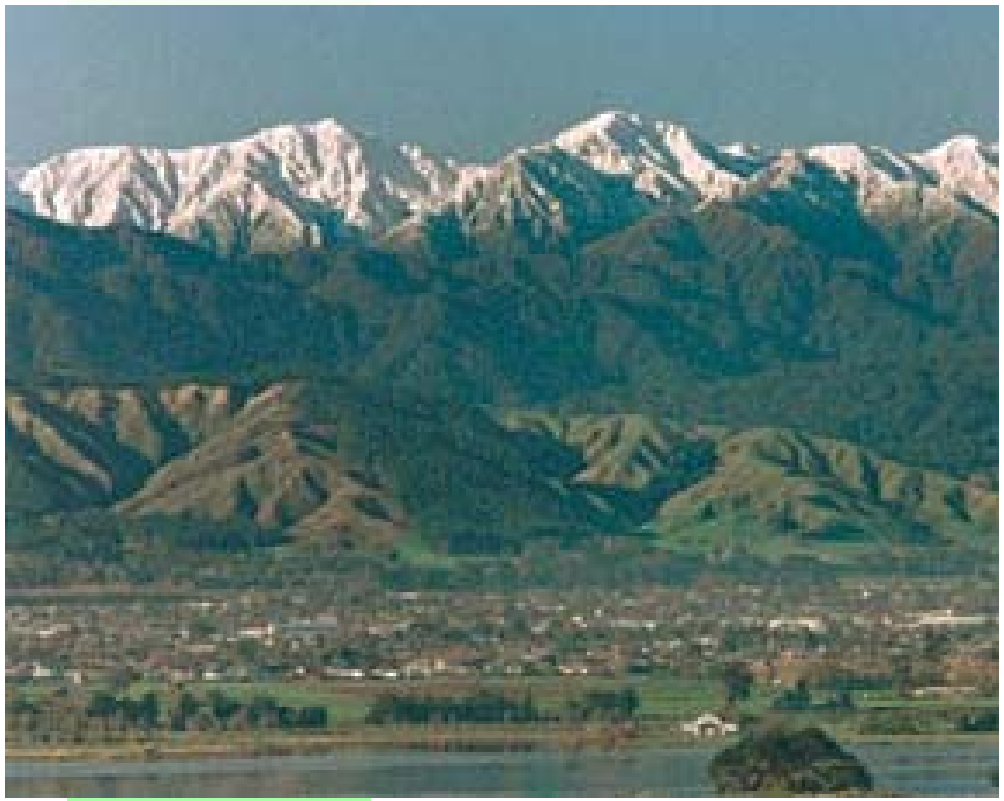
Total Code 3.1415×10^{9} function points	100%
C 1.1×10^{9} function points	35%
Ratfor 2.0415×10^{9} function points	65%
Test Material 2.718×10^{6} bytes	100%
Data 2.69×10^{6} bytes	99%
Executables 27.18×10^{3} bytes	1%
Documentation 12 058 pages	100%
Tutorial 3 971 pages	33%
Reference 6 233 pages	52%
Design & Specification 1 854 pages	15%

Libraries: Sun Java 1.5 runtime, Sun J2EE 1.2.2, Jakarta log4j 1.5, Jakarta Commons 2.1, Jakarta Struts 2.0, Harold XOM 1.1rc4, Hunter JDOMv1

Compiled with gcc (GCC) 3.3.1

Stripped of all symbols and relocation information.

Tools Useful in Quality “Plains”



Tararua mountains and the Horowhenua region, New Zealand
Swazi Apparel Limited www.swazi.co.nz used with permission

- **Tools are not enough to achieve the highest “peaks” of quality.**
- **In the “plains” of typical quality, tools can help.**
- **If code is adrift in a “sea” of chaos, train developers.**