Foundations of Software Assurance

Paul E. Black
Software Quality Group
Software and Systems Division

16 June 2016
Outline

- Software Assurance Reference Dataset (SARD)
- Bugs Framework (BF)
Software Assurance Reference Dataset (SARD)

http://samate.nist.gov/SARD/
Software Assurance Reference Dataset (SARD)

- Public repository for software assurance test cases with known vulnerabilities
- Over 140 000 cases in C, C++, Java, PHP, C#, and Python
- Contributions from NSA/CAS, IARPA, Fortify, TELECOM Nancy, Defence R&D Canada, Klocwork, MIT Lincoln Laboratory, Praxis, Toyota, Secure Software, etc.

http://samate.nist.gov/SARD/
What is Static Analysis?

Java, C, C++, ... binary
What is Static Analysis?

- Examine source code or binary for weaknesses, adherence to guidelines, etc.
How to Test Static Analyzers?

1. Programs with known bugs
2. Static Analyzer
3. Weaknesses & Vulnerabilities
4. Known Weaknesses & Vulnerabilities
Characteristics of Test Cases

Production Code

programs with known bugs
Characteristics of Test Cases

Production Code

Known Bugs

programs with known bugs
Characteristics of Test Cases

Programs with known bugs

Production Code

Statistically Significant

Known Bugs

Perfect Test Suite
Characteristics of Test Cases

- **Approximations**
  - Collect millions of tool warnings for open source software from SATE.
  - Manually analyze hundreds of reported bugs (CVEs) to establish ground truth.
  - Publish Juliet test suite: hundreds of thousands of synthetic test cases with known bugs.
SARD Content

- Contributions also from Kratkiewicz, MIT Lincoln Laboratory, Praxis, etc.
- NSA Juliet 1.2 - over 86,000 small, synthetic test cases in C, C++, and Java, covering 150 bug classes
- IARPA STONESOUP Phase 3 - 15,000 cases based on 12 web apps with injected bugs from 25 classes
- 1276 test cases from Toyota
- Test cases from Static Analysis Tool Exposition (SATE)
- 2000 PHP cases developed at TELECOM Nancy
Other SARD 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
Outline

- Software Assurance Reference Dataset (SARD)
- Bugs Framework (BF)

http://samate.nist.gov/BF/
The Bugs Framework (BF) is a precise descriptive language for bugs.
Precise Medical Language

- Medical professionals have terms to precisely name muscles, bones, organs, conditions, diseases, etc.

Figure 2: Computed tomography of a comatose patient with a left temporal epidural haematoma, right parenchymal temporal lobe haematoma, and a right convexity subdural haematoma before and after craniotomy and evacuation of haematomas
Current Bug Descriptions Have Problems

- **Common Weakness Enumeration (CWE)**
  - Definitions are imprecise and inconsistent.
  - Coarse grained: bundling attributes, attacks, etc.
  - Uneven coverage: some combinations not given all.

- **Software Fault Patterns (SFP)**
  - Does not include upstream causes or consequences.
  - Based solely on CWEs.

- **Semantic Templates**
  - Does not distinguish many types of fault, weakness, location, or consequence.
  - Only cover two classes.
What is the Bugs Framework?

• It is a set of classes of bugs.
• Each bug class has
  – Causes
  – Attributes of a fault
  – Consequences
• Causes and consequences are directed graphs.
• BF uses precise terminology.
Bugs Framework Classes

- **Injection (INJ), e.g.**
  - SQL injection
  - OS injection

- **Control of Interaction Frequency (CIF), e.g.**
  - Limit number of login attempts
  - Only one vote per voter

- **Information Exposure (IEX), e.g.**
  - Password leak

- **Buffer Overflow (BOF)**
Buffer Overflow: Attributes
Buffer Overflow: Attributes

- Access:
  - Read, Write.
Buffer Overflow: Attributes

- **Access:**
  - Read, Write.
- **Boundary:**
  - Below (before, under, or lower), Above (after, over, or upper).
Buffer Overflow: Attributes

- Access:
  - Read, Write.
- Boundary:
  - Below (before, under, or lower), Above (after, over, or upper).
- Location:
  - Heap, Stack, BSS (uninitialized data), Data (initialized), Code (text).
Buffer Overflow: Attributes

- **Access:**
  - Read, Write.
- **Boundary:**
  - Below (before, under, or lower), Above (after, over, or upper).
- **Location:**
  - Heap, Stack, BSS (uninitialized data), Data (initialized), Code (text).
- **Magnitude (how far outside):**
  - Small (just barely outside), Far (e.g. 4000).
Buffer Overflow: Attributes

- **Access:**
  - Read, Write.

- **Boundary:**
  - Below (before, under, or lower), Above (after, over, or upper).

- **Location:**
  - Heap, Stack, BSS (uninitialized data), Data (initialized), Code (text).

- **Magnitude (how far outside):**
  - Small (just barely outside), Far (e.g. 4000).

- **Data Size (how much is outside):**
  - Little, Huge.
Buffer Overflow: Attributes

- **Access:**
  - Read, Write.

- **Boundary:**
  - Below (before, under, or lower), Above (after, over, or upper).

- **Location:**
  - Heap, Stack, BSS (uninitialized data), Data (initialized), Code (text).

- **Magnitude (how far outside):**
  - Small (just barely outside), Far (e.g. 4000).

- **Data Size (how much is outside):**
  - Little, Huge.

- **Reach (one-by-one or arbitrary):**
  - Continuous, Discrete.
Buffer Overflow: Causes

- **Causes**
  - Input Not Checked Properly
  - Data Exceeds Array
    - Array Too Small
    - Too Much Data
  - Wrong Index / Pointer Out of Range
  - Incorrect Conversion
  - No NULL Termination

- **Attributes**
  - **Access:**
    - Read
    - Write
  - **Boundary:**
    - Below
    - Above
  - **Location:**
    - Heap
    - Stack
  - **Magnitude:**
    - Small
    - Far
  - **Data Size:**
    - Little
    - Huge
  - **Reach:**
    - Continuous
    - Discrete
Buffer Overflow: Consequences

Attributes
- Access:
  - Read
  - Write
- Boundary:
  - Below
  - Above
- Location:
  - Heap
  - Stack
- Magnitude:
  - Small
  - Far
- Data Size:
  - Little
  - Huge
- Reach:
  - Continuous
  - Discrete

Consequences
- Information Exposure
- Information Change/Loss
- Altered Control Flow
- Incorrect Results
- Program Crash
- System Crash
- Resource Exhaustion
- Denial Of Service
- Arbitrary Code Execution
- System Crash
- Program Crash
- Incorrect Results
- Resource Exhaustion
- Denial Of Service
- Arbitrary Code Execution
What is BF Good For?

- Precisely explain why techniques work in some cases and not others.
- More clearly describe vulnerabilities (e.g. Heartbleed, Shellshock, and Ghost).
- Help programmers write better code, because they understand weaknesses more clearly.
- Accurately state the classes of bugs that software assurance tools cover (and do not cover).
Example 1: BF Explains Techniques

• Canaries
  – A canary is extra memory above and below an array with unusual values, e.g., 0xDEADBEEF
  – Useful with attributes
    • Write Access
    • Small Magnitude

• Address Space Layout Randomization (ASLR)
  – Allocate arrays randomly about memory
  – Useful with attributes
    • Heap Location
    • Stack Location - limited
Example 2: Heartbleed

from http://xkcd.com/1354/
Heartbleed buffer overflow is:
- caused by *Data Exceeds Array*, specifically *Too Much Data*
- because of *Input not Checked Properly*
- where there was a *Read* that was *After* the end, *Far* outside
- in a *Continuous* read of a *Huge* number of bytes
- from an array in the *Heap*
- that may be exploited for *Information Exposure*
- when enabled by *Sensitive Information uncleared before release (CWE-226)*.

“The (1) TLS and (2) DTLS implementations … do not properly handle Heartbeat Extension packets, which allows remote attackers to obtain sensitive information from process memory via crafted packets that trigger a buffer over-read, as demonstrated by reading private keys, …” (CVE-2014-0160)
Example 2: Heartbleed

- Sensitive Info Uncleared Before Release
- Input Not Checked Properly
- Data Exceeds Array
  - Array Too Small
  - Too Much Data
- No NULL Termination
- Wrong Index / Pointer Out of Range
- Incorrect Calculation
  - Missing Factor
  - Integer Coercion
  - Integer Overflow
  - Wrap-around
  - Integer Underflow
  - Off By One
- Incorrect Conversion

Access:
- Read
- Write

Boundary:
- Below
- Above

Location:
- Heap
- Stack

Magnitude:
- Small
- Far

Data Size:
- Little
- Huge

Reach:
- Continuous
- Discrete

Information Change/Loss
- Altered Control Flow
- Incorrect Results
- Resource Exhaustion
- System Crash
- Program Crash
- Denial Of Service
- Arbitrary Code Execution