SARD: A Software Assurance Reference Dataset

Paul E. Black

SAMATE Project

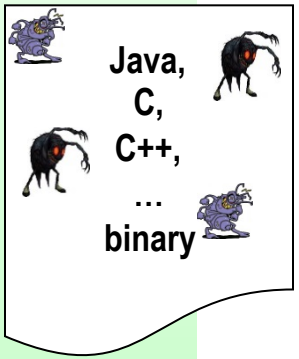
Software Quality Group

Software & Systems Division

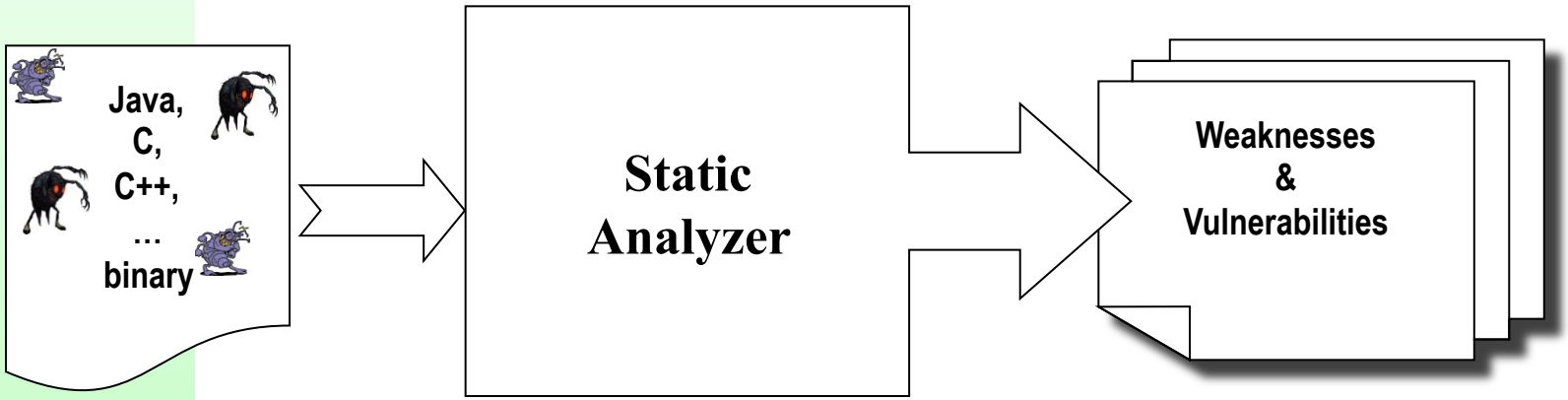
NIST

Certain trade names and company products are mentioned in the text or identified. In no case does such identification imply recommendation or endorsement by the National Institute of Standards and Technology (NIST), nor does it imply that the products are necessarily the best available for the purpose.

What is Static Analysis?

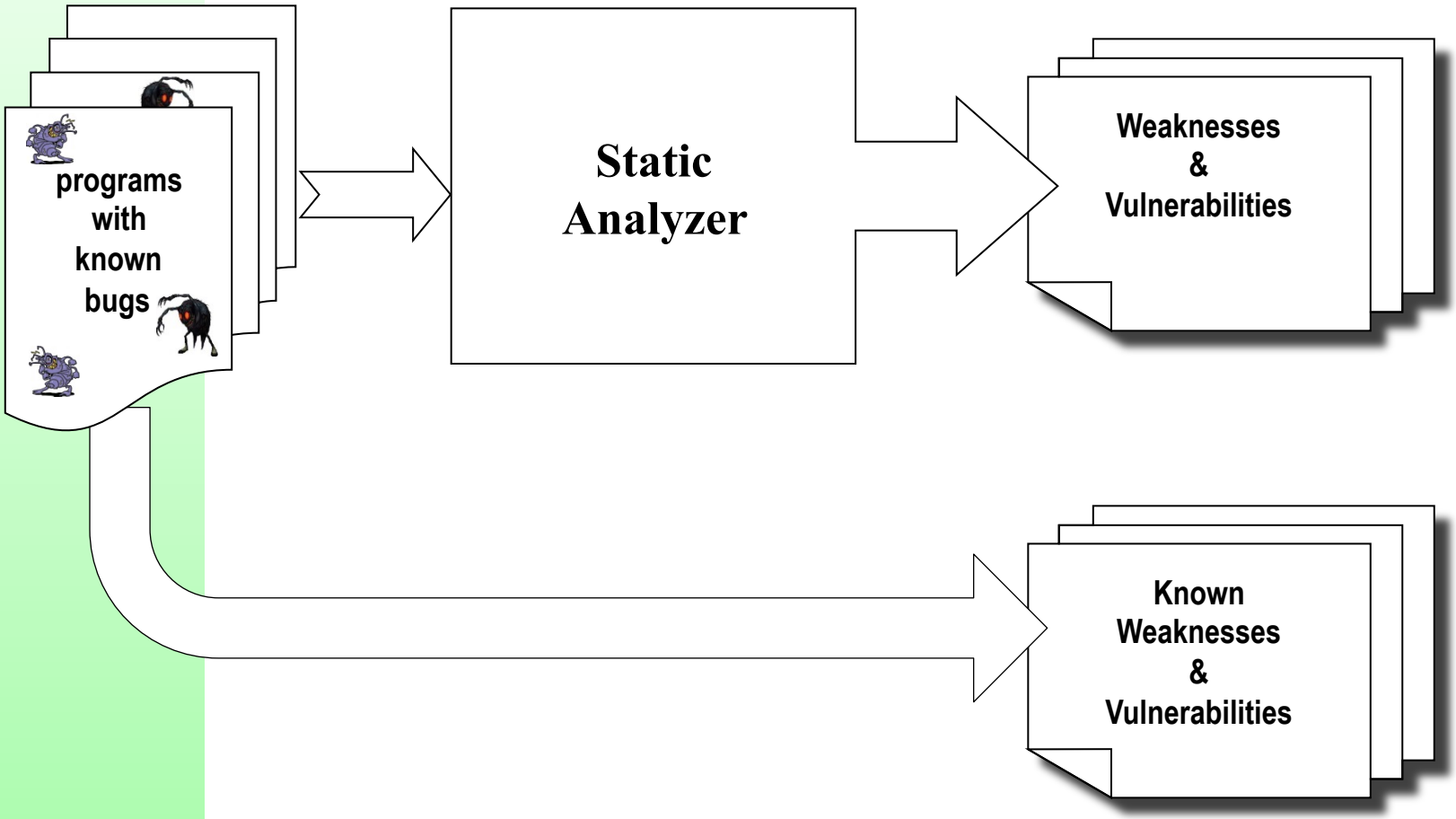


What is Static Analysis?



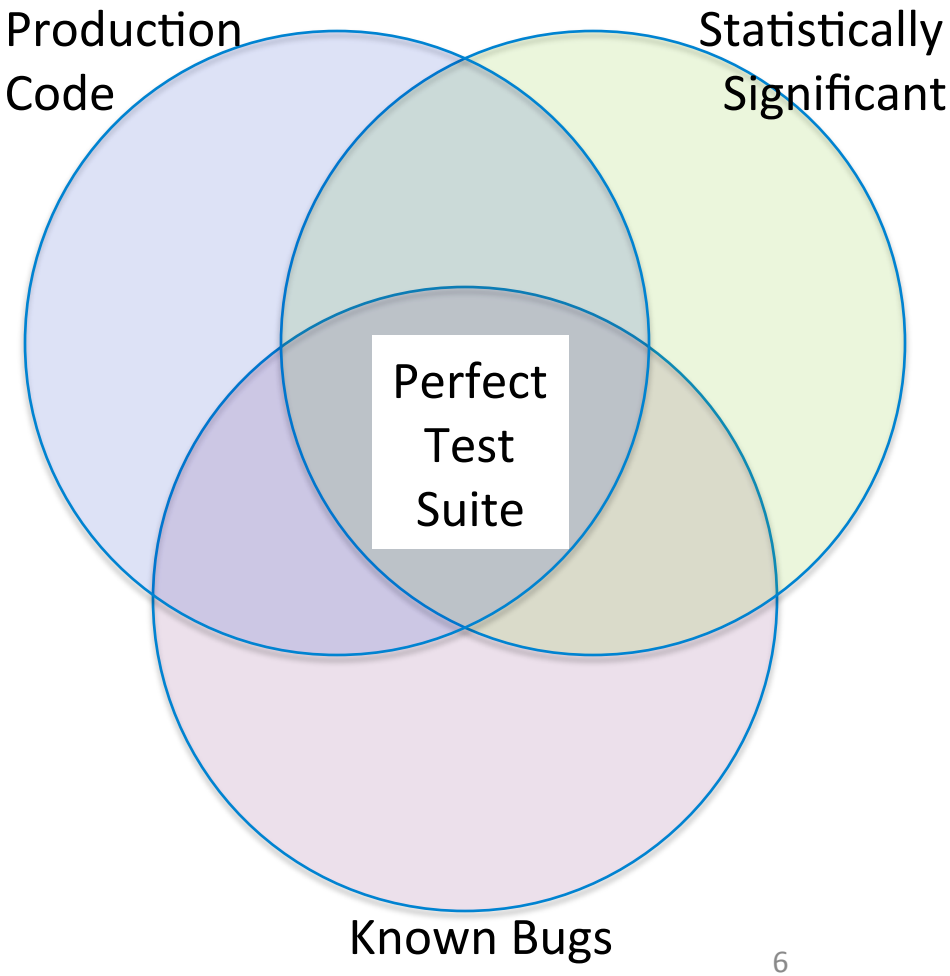
- Examine source code or binary for weaknesses, adherence to guidelines, etc.
- Level of formality may vary from program "proofs" to heuristics
- Level of automation may vary from analysis assistant to fully automated

Testing Static Analysis Tools



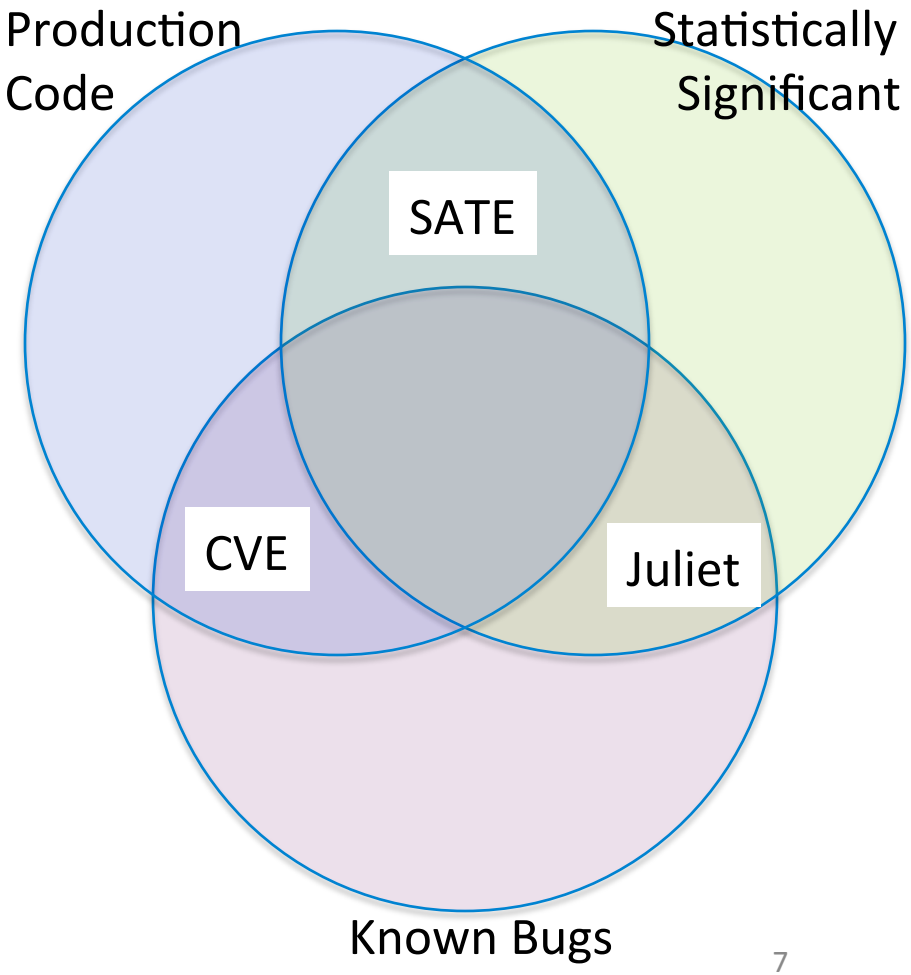
Three Desired Characteristics of Test Suites

- **Needs**
  - **Test cases applicable to production code**
  - **Statistically significant number of test cases**
  - **Test cases with ground truth: known bugs**
- **Objective:**
  - **Collect and develop test cases with those characteristics**



Three Desired Characteristics of Test Suites

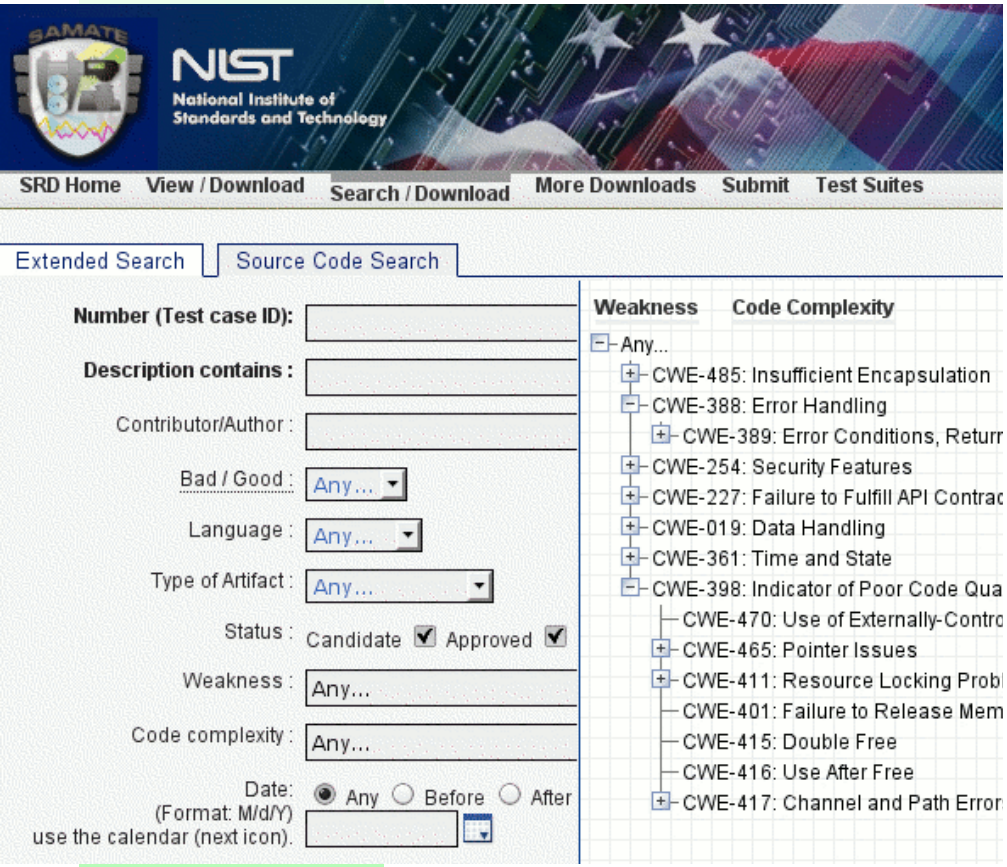
- **Achievements**
  - **Collect millions of tool warnings for open source software from SATE**
  - **Manually analyze hundreds of reported bugs (CVEs) in open source software to establish ground truth**
  - **Juliet test suite: hundreds of thousands of synthetic test cases with known bugs**



Material to Properly Test Tools

- **Static analysis**
- **Dynamic bug detection**

Software Assurance Reference Dataset (SARD)



Need:

- **Suites of programs with known bugs to calibrate software assurance tools**

Objective:

- **Collect and develop sets of programs with known bugs in various languages, with bugs of various classes, and bugs woven into various code structures**

http://samate.nist.gov/SARD/

Software Assurance Reference Dataset (SARD)

- Over 89 000 cases in C, C++, Java, C#, and PHP
- Contributions also from Fortify, Defence R&D Canada, Klocwork, Kratkiewicz, MIT Lincoln Laboratory, Secure Software, 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 bug from 25 classes
- 2000 PHP cases developed at TELECOM Nancy
- Users can search and download by language, weakness, size, content, etc.
- Test cases from Static Analysis Tool Exposition (SATE)

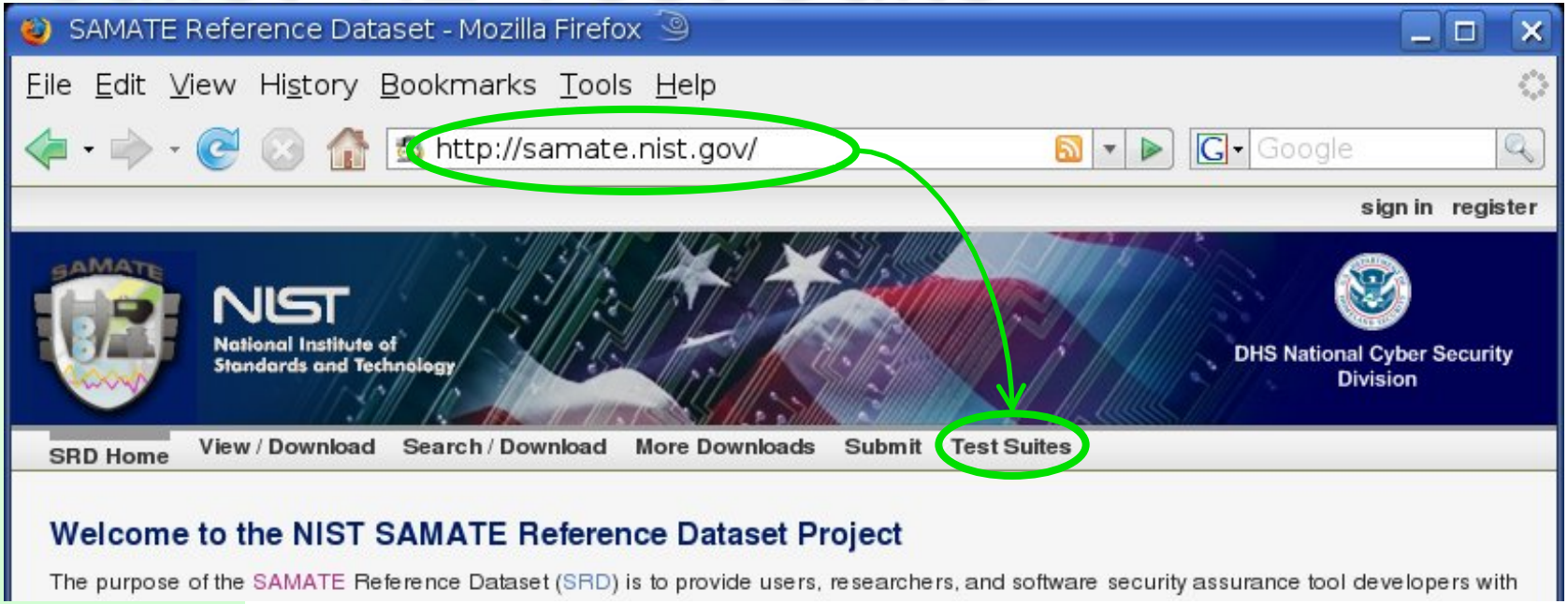
TELECOM Nancy
Ingénieurs du numérique • Inspiring your digital future

IARPA
BE THE FUTURE

LINCOLN LABORATORY
MASSACHUSETTS INSTITUTE OF TECHNOLOGY

NATIONAL SECURITY AGENCY
UNITED STATES OF AMERICA

Juliet 1.2 Test Suite



- **86 864 small C/C++ and Java programs for almost two hundred weakness classes**
- **Each case is one or two pages of code**
- **Described in IEEE Computer, Oct 2012**

IARPA STONESOUP Phase 3 cases

- **7770 cases in Java and C**
- **Real programs with flaw inserted. Each case has inputs to trigger flaw and “safe” inputs**
- **Each case has inputs triggering the vulnerability, as well as “safe” inputs**
- **Cover weaknesses in Integer Overflow, Tainted Data, Command Injection, Buffer Overflow, Null Pointer, etc.**

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

Common Nomenclature

Common Weakness Enumeration (CWE)

- A "dictionary" of every *class* of bug or flaw in software
- More than 600 distinct classes, e.g., buffer overflow, directory traversal, OS injection, race condition, cross-site scripting, hard-coded password, and insecure random numbers

http://cwe.mitre.org/

Common Vulnerability Enumeration (CVE)

- A list of *instances* of security vulnerabilities in software
- More than 9000 CVEs were assigned in 2014
  Heartbleed is CVE-2014-0160
- NIST's National Vulnerability Database (NVD) has fixes, severity ratings, etc. for CVEs

https://cve.mitre.org/

Common Weakness Enumeration (CWE) is a Mess

- **CWE is widely used - by far the best dictionary of software weaknesses. Many tools, projects, etc. are based on CWE.**
- **But definitions are imprecise and inconsistent.**
- **CWEs are "coarse grained": they bundle lots of stuff, like consequences and likely attacks.**
- **The coverage is uneven, with some combinations well represented and others not represented at all.**
- **No mobile weaknesses, eg., battery drain, physical sensors (GPS, gyro, microphone, hi-res camera), unencrypted wireless communication, etc.**

Definitions are Imprecise

- **CWE-119: Improper Restriction of Operations within the Bounds of a Memory Buffer:**

  "The software performs operations on a memory buffer, but it can read from or write to a memory location that is outside of the intended boundary of the buffer."

  - ***Note that "read from or write to a memory location" is not tied to the buffer!***

Overflow Has Gaps in Coverage

- **CWE-124: Buffer Underwrite (’Buffer Underflow') and CWE-120: Buffer Copy without Checking Size of Input ('Classic Buffer Overflow') vs.**
- **CWE-121: Stack-based Buffer Overflow and CWE-122: Heap-based Buffer Overflow**

- **CWE-127: Buffer Under-read and CWE-126: Buffer Over-read**
- ***but no read-stack and read-heap versions*.**

… and a buncha' others, too

- **CWE-123: Write-what-where Condition**
- **CWE-125: Out-of-bounds Read**
- **CWE-787: Out-of-bounds Write**
- **CWE-786: Access of Memory Location Before Start of Buffer**
- **CWE-788: Access of Memory Location After End of Buffer**
- **CWE-805: Buffer Access with Incorrect Length Value**
- **CWE-823: Use of Out-of-range Pointer Offset**

Path Traversal is too Detailed

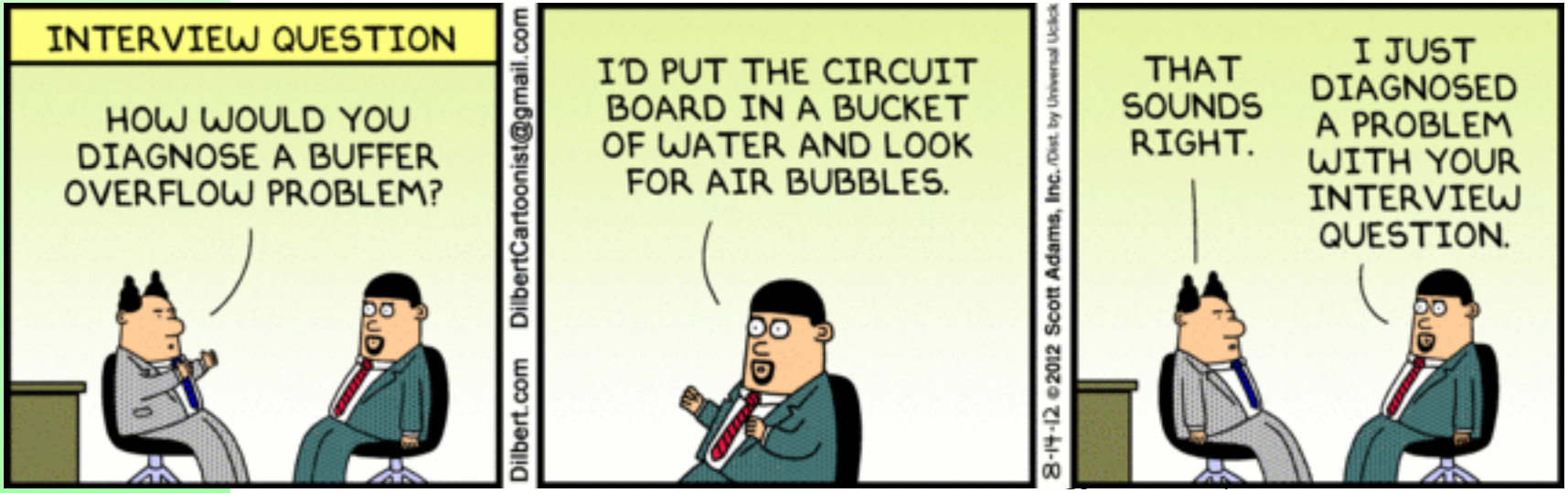
- **CWE-23: Relative Path Traversal**
- **CWE-24: Path Traversal: '../filedir'**
- **CWE-25: Path Traversal: '/../filedir'**
- **CWE-26: Path Traversal: '/dir/../filename'**
- **CWE-27: Path Traversal: 'dir/../../filename'**
- **CWE-28: Path Traversal: '..\filedir'**
- **CWE-29: Path Traversal: '\..\filename'**
- **CWE-30: Path Traversal: '\dir\..\filename'**
- **CWE-31: Path Traversal: 'dir\..\..\filename'**
- **CWE-32: Path Traversal: '...' (Triple Dot)**
- **CWE-33: Path Traversal: '....' (Multiple Dot)**
- **CWE-34: Path Traversal: '..../'**
- **CWE-35: Path Traversal: '.../.../'**

Other Bug Descriptions Have Problems, Too.

- **Software Fault Patterns (SFP)**
  - **“factor” weaknesses into parameters, but**
  - **don’t include upstream causes or consequences,**
  - **and are based solely on CWEs.**
- **Semantic Templates**
  - **collect CWEs into four general areas**
    - **Software-fault**
    - **Weakness**
    - **Resource/Location**
    - **Consequences**
  - **but are guides to aid human comprehension.**

We Need Better Vocabulary

- **Finer grained, common vocabulary to describe bugs**
  - **Common Weakness Enumeration (CWE) is widely-used, but does not match well the classes that tools report. Tools' classes are precise, but are hard to match to other tools.**



Precise Medical Vocabulary

- Medical professionals have terms to precisely name muscles, bones, organs, conditions, diseases, and so forth.

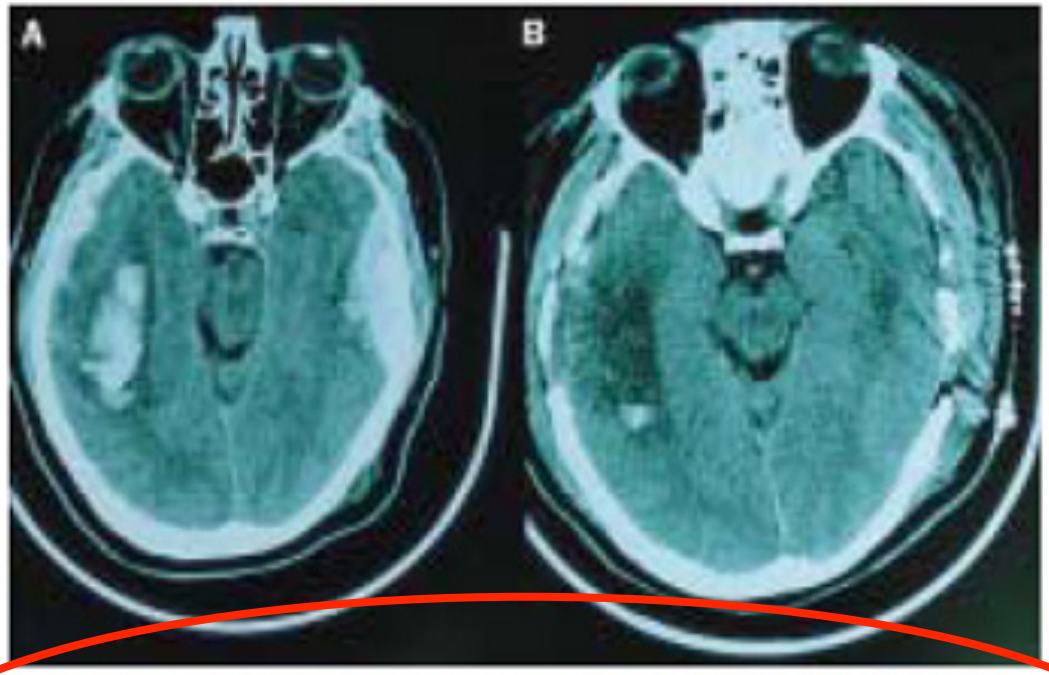
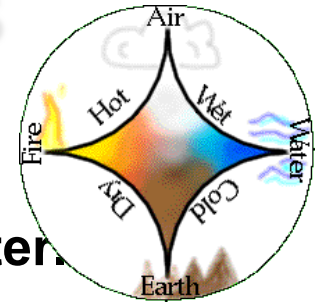
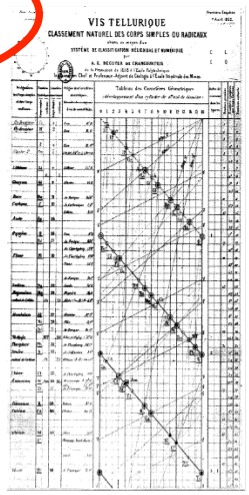


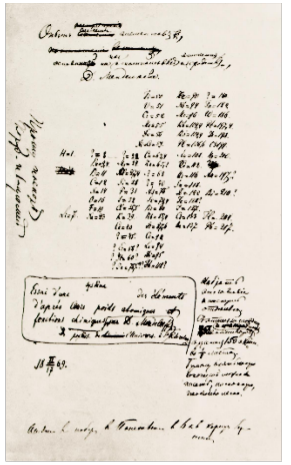
Figure 2: **Computed tomography of a comatose patient with a left temporal epidural haematoma, right parenchymal temporal lobe haematoma, and a right convexity subdural haematoma before and after craniotomy and evacuation of haematomas**

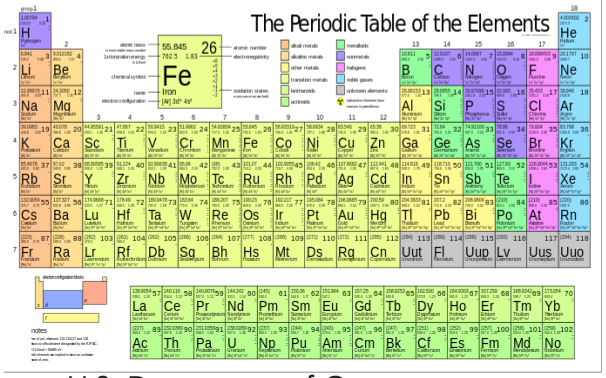
Periodic Table Took Centuries



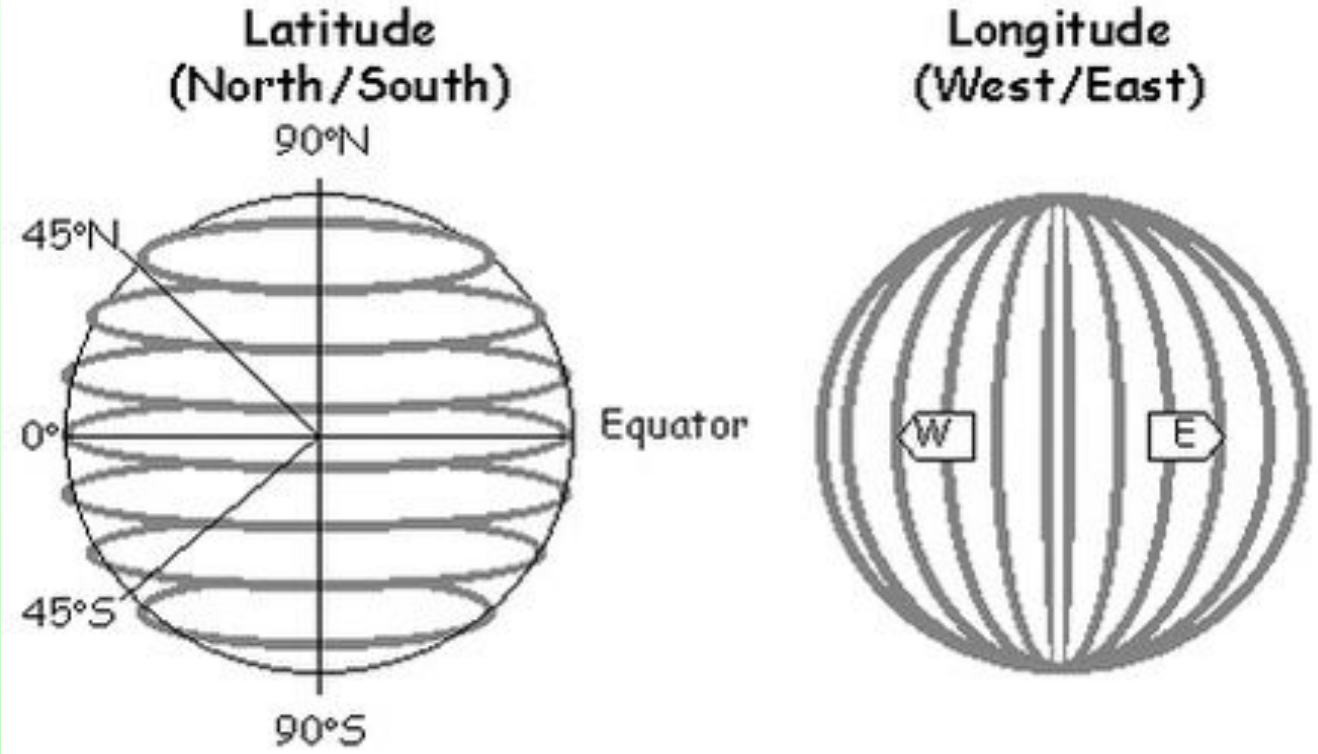
- Greeks used the terms *element* and *atom*.
- Aristotle: everything is a mix of Earth, Fire, Air, or Water.
- Alchemists in the Middle Ages cataloged materials like alcohol, sulfur, mercury, and salt.
- Lavoisier listed 33 elements and distinguished metals and non-metals.
  - including oxygen, nitrogen, hydrogen, phosphorus, mercury, zinc, sulfur, *light*, and *caloric*.
- Dalton realized "atoms of same element are identical in all respects, particularly weight."
- Mendeleev's table embodied centuries of knowledge that reflects atomic structure and forecast properties of missing elements.





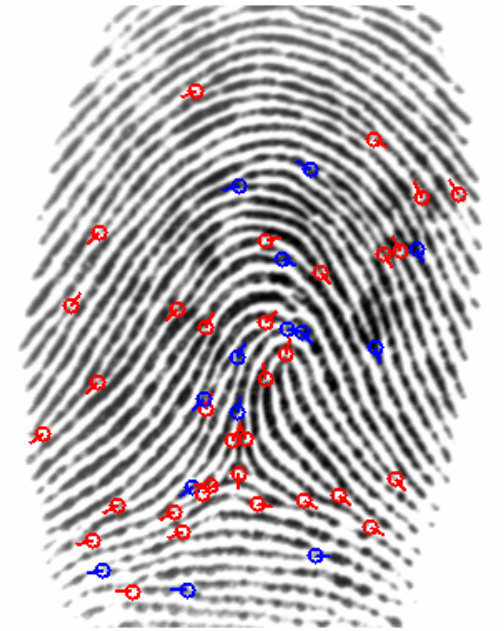


Specify Location with Latitude, Longitude, and Elevation



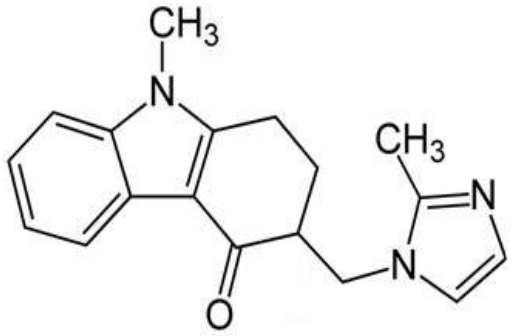
Fingerprints

- **Classified as loop, whorl, or arch.**
- **Retrieved by minutia**



Chemists Have Detailed Systems to Describe Chemicals

Zofran ODT is: $C_{18}H_{19}N_3O$



(±) 1, 2, 3, 9-tetrahydro-9-methyl-3-[(2-methyl-1H-imidazol-1-yl)methyl]-4H-carbazol-4-one

Integers Have Prime Factors

$$6 = 2 \times 3$$

$$70 = 2 \times 5 \times 7$$

$$47,298,756 = 2 \times 2 \times 7 \times 641 \times 1471 \times 1657$$

Our vision is to have
a precise descriptive language for bugs
organized in a “natural” way.

(e.g., vocabulary, grammar, taxonomy, ontology, etc.)

Outline

- The "Science" of Weaknesses
- **Our Nomenclature**
- Examples of Applying Our Approach
- Using This

We Start With Buffer Overflow

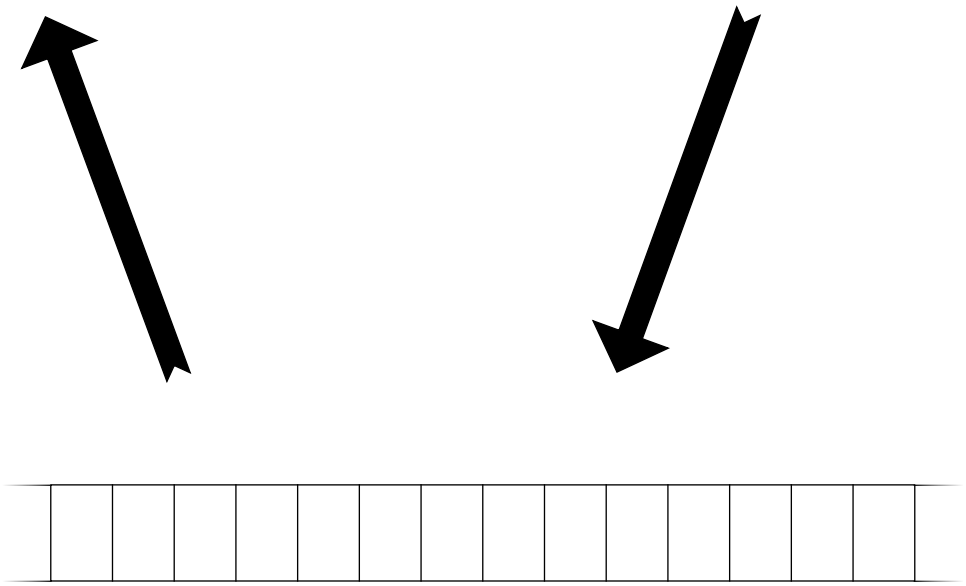
- **Our Definition:**
  The software can access through a buffer a memory location that is not allocated to that buffer.

- **Clearer than CWE-119: Improper Restriction of Operations within the Bounds of a Memory Buffer:**
  "The software performs operations on a memory buffer, but it can read from or write to a memory location that is outside of the intended boundary of the buffer."

Buffer Overflow: Attributes

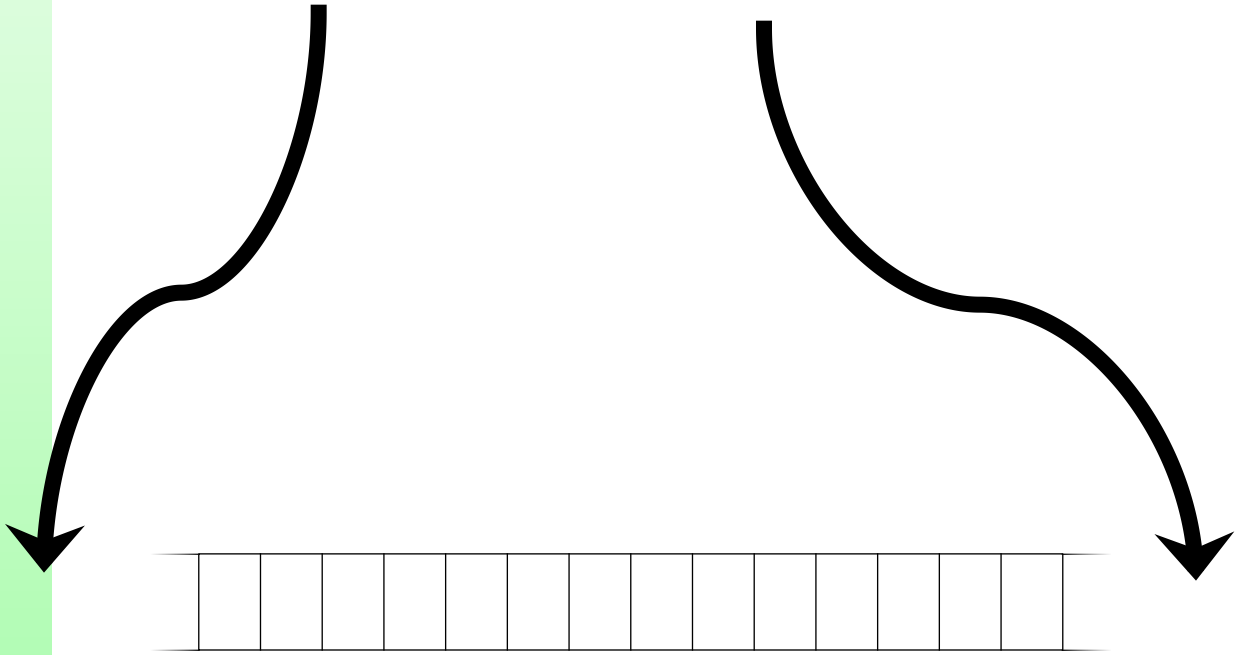
Buffer Overflow: Attributes

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



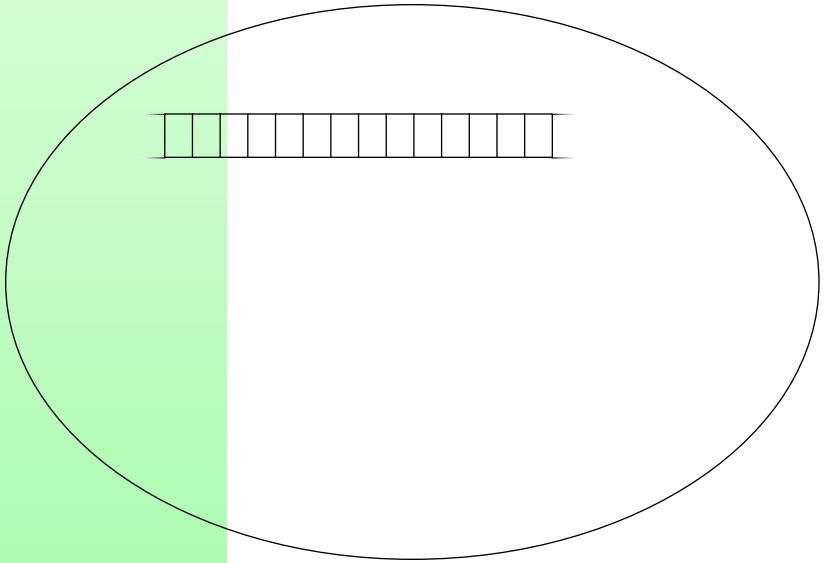
Buffer Overflow: Attributes

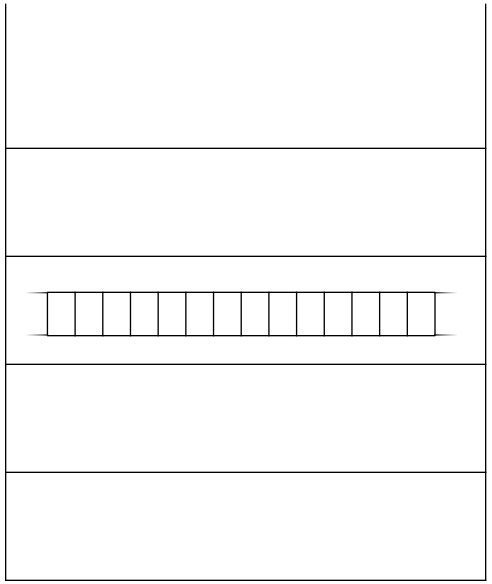
- **Access:**
  - **Read, Write.**
- **Side:**
  - **Below (before, under, or lower), Above (after, over, or upper).**



Buffer Overflow: Attributes

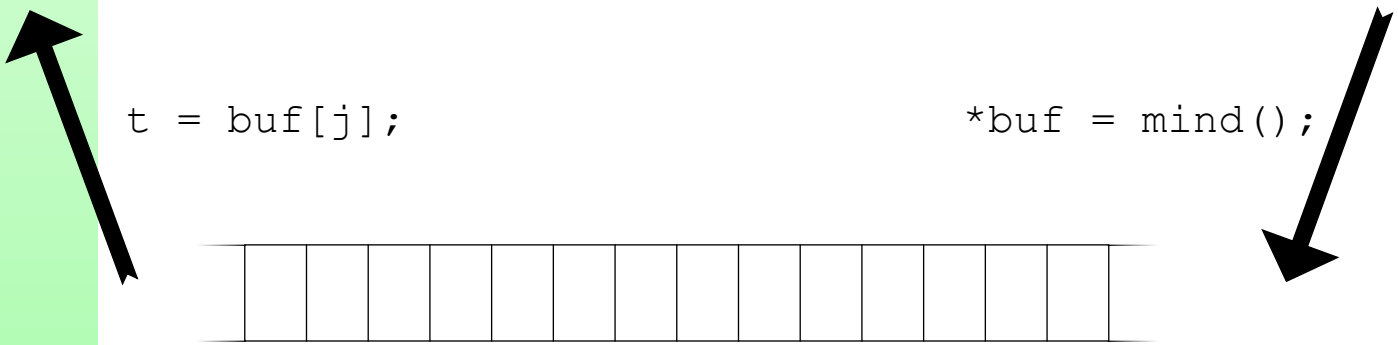
- **Access:**
  - **Read, Write.**
- **Side:**
  - **Below (before, under, or lower), Above (after, over, or upper).**
- **Segment (memory area):**
  - **Heap, Stack, BSS (uninitialized data), Data (initialized), Code (text).**





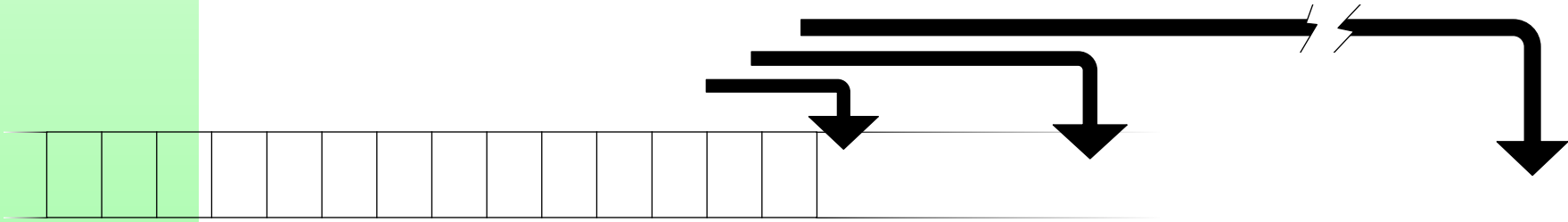
Buffer Overflow: Attributes

- **Access:**
  - **Read, Write.**
- **Side:**
  - **Below (before, under, or lower), Above (after, over, or upper).**
- **Segment (memory area):**
  - **Heap, Stack, BSS (uninitialized data), Data (initialized), Code (text).**
- **Method:**
  - **Indexed, (bare) Pointer.**



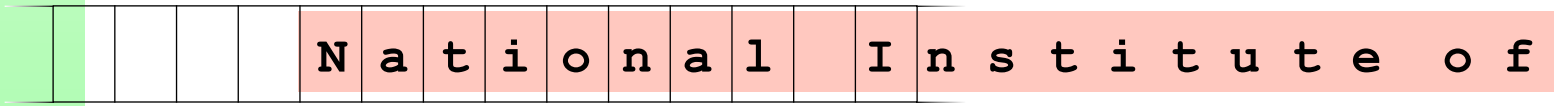
Buffer Overflow: Attributes

- **Access:**
  - **Read, Write.**
- **Side:**
  - **Below (before, under, or lower), Above (after, over, or upper).**
- **Segment (memory area):**
  - **Heap, Stack, BSS (uninitialized data), Data (initialized), Code (text).**
- **Method:**
  - **Indexed, (bare) Pointer.**
- **Magnitude (how far outside):**
  - **Minimal (just barely outside), Moderate, Far (e.g. 4000).**

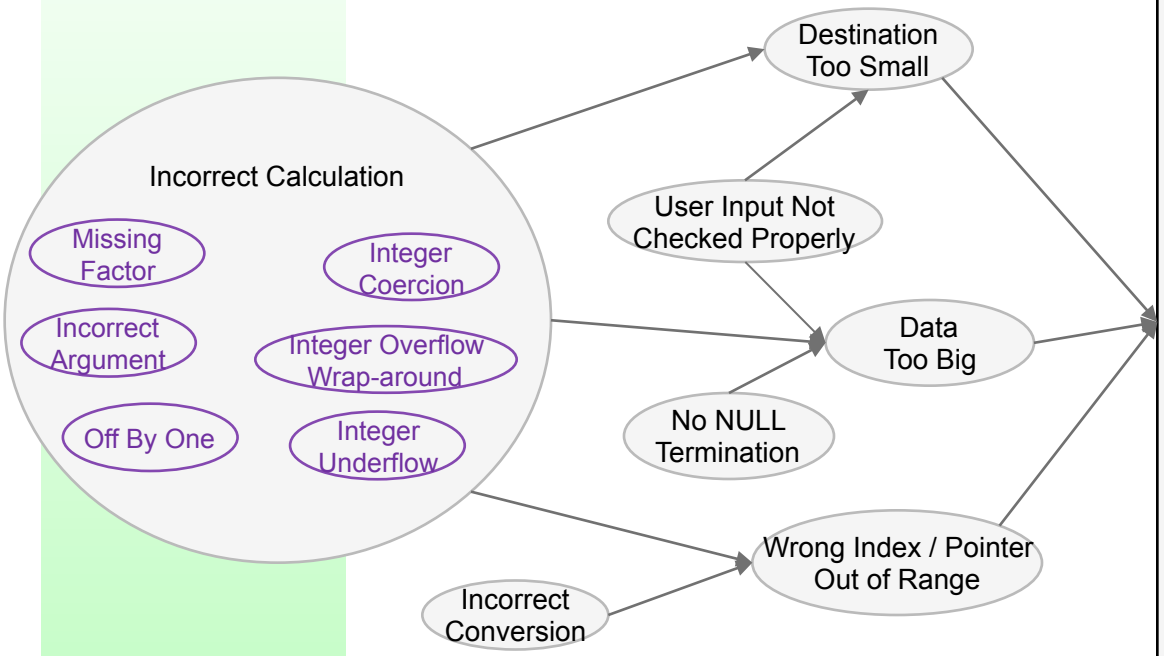


Buffer Overflow: Attributes

- **Access:**
  - **Read, Write.**
- **Side:**
  - **Below (before, under, or lower), Above (after, over, or upper).**
- **Segment (memory area):**
  - **Heap, Stack, BSS (uninitialized data), Data (initialized), Code (text).**
- **Method:**
  - **Indexed, (bare) Pointer.**
- **Magnitude (how far outside):**
  - **Minimal (just barely outside), Moderate, Far (e.g. 4000).**
- **Data Size (how much is outside):**
  - **Minimal, Some (e.g. half dozen), Gazillion.**



Buffer Overflow: Causes



Buffer Overflow

Attributes:

- Access:
  - ✓*Read*, *Write*.
- Side:
  - ✓*Below* (before or under), *Above* (after or over)
- Segment (memory area):
  - ✓*Heap*, *Stack*, *BSS*, *Data* (initialized), *Code* (text)
- Method:
  - ✓*Indexed*, (bare) *Pointer*.
- Magnitude (how far outside):
  - ✓*Minimal* (just barely), *Moderate*, *Far* (e.g. 4000).
- Data Size (how much data) :
  - ✓*Minimal*, *Some*, *Gazillion*.

The graph of causes shows:

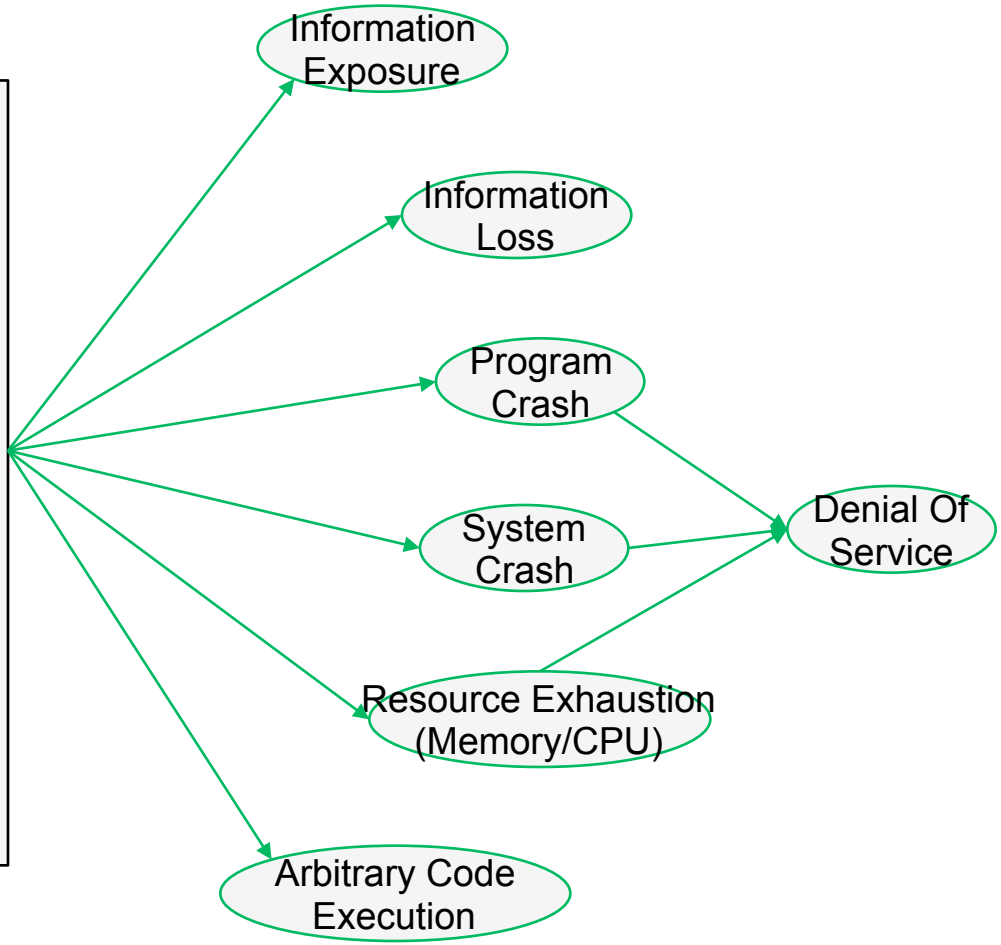
- There are only 3 proximate causes of buffer overflows:
  - Destination is too small
  - Data is too big
  - Wrong index / pointer out of range.
- Those 3 have preceding causes that may lead to them.

Buffer Overflow: Consequences

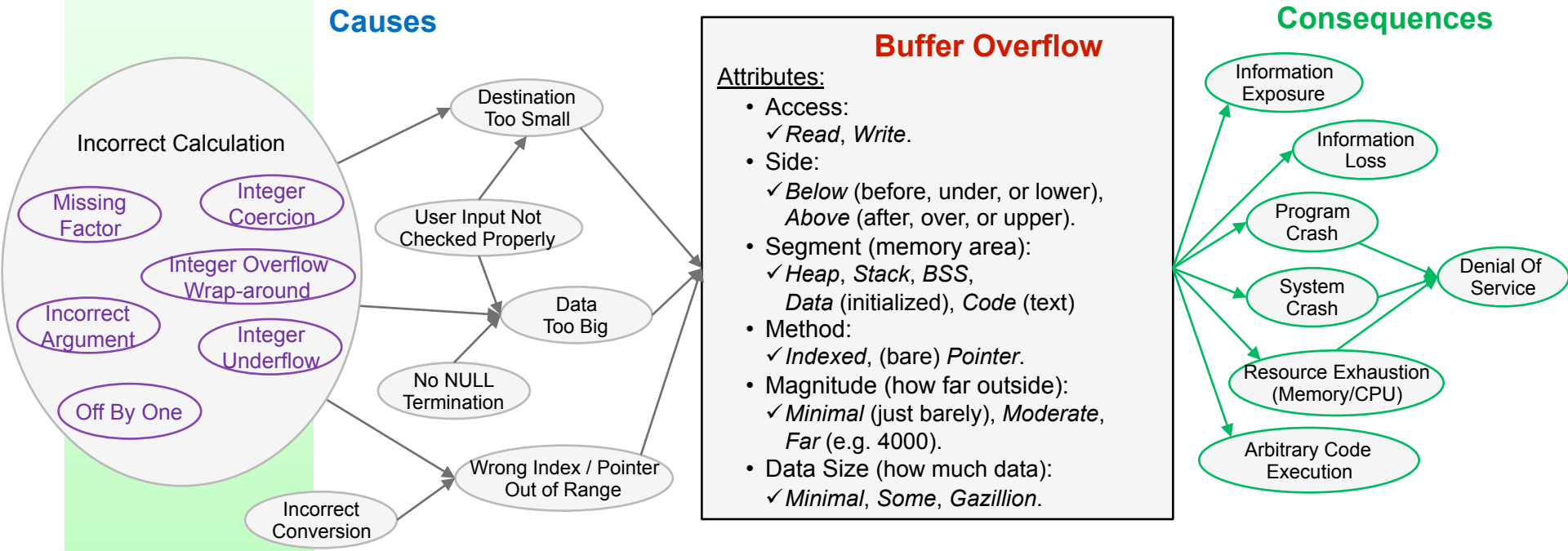
Buffer Overflow

Attributes:

- Access:
  - ✓*Read*, *Write*.
- Side:
  - ✓*Below* (before, under, or lower), *Above* (after, over, or upper).
- Segment (memory area):
  - ✓*Heap*, *Stack*, *BSS*, *Data* (initialized), *Code* (text)
- Method:
  - ✓*Indexed*, (bare) *Pointer*.
- Magnitude (how far outside):
  - ✓*Minimal* (just barely), *Moderate*, *Far* (e.g. 4000).
- Data Size (how much data) :
  - ✓*Minimal*, *Some*, *Gazillion*.



Buffer Overflow: Causes, Attributes, and Consequences



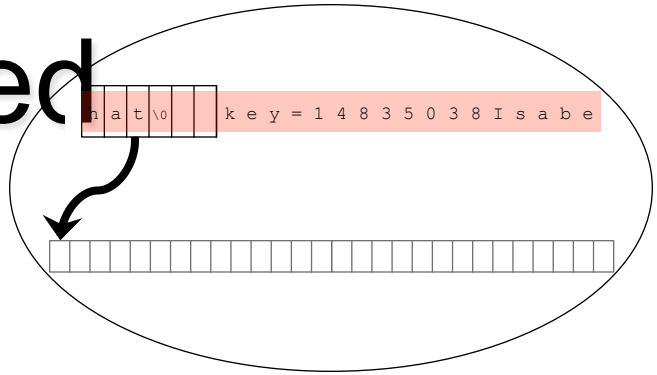
The graph of causes shows:

- There are only 3 proximate causes of buffer overflows:
  - Destination is too small
  - Data is too big
  - Wrong index / pointer out of range.
- Those 3 have preceding causes that may lead to them.

Outline

- The "Science" of Weaknesses
- Our Nomenclature
- **Examples of Applying Our Approach**
- Using This

Example 1: Heartbleed CVE-2014-0160

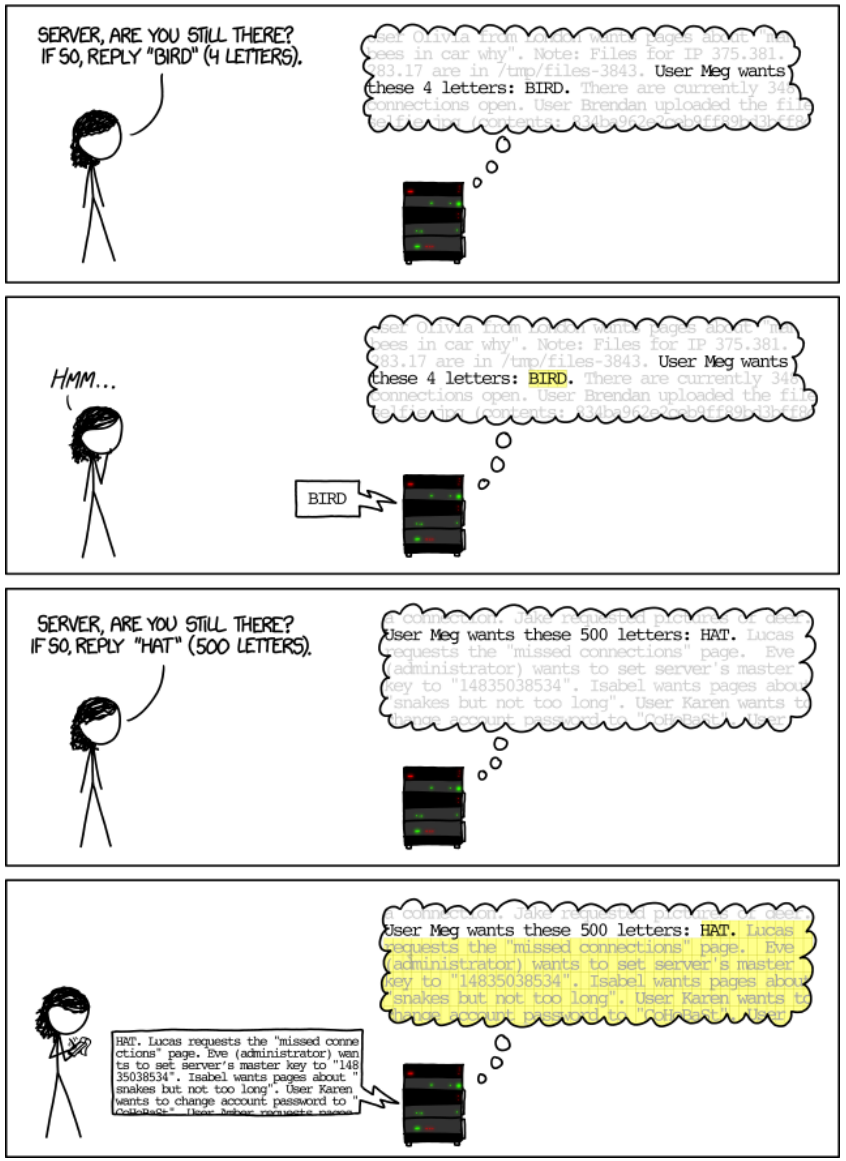


Heartbleed buffer overflow is:

- **caused by *Data Too Big***
- **because of *User Input not Checked Properly***
- **where there was a *Read* that was *After* the end, *Far* outside**
- **reading a *Gazillion* bytes**
- **from a buffer in the *Heap***
- **that may be exploited for *Information Exposure***
- **when enabled by *Sensitive Information Uncleared Before Release (CWE-226)*.**

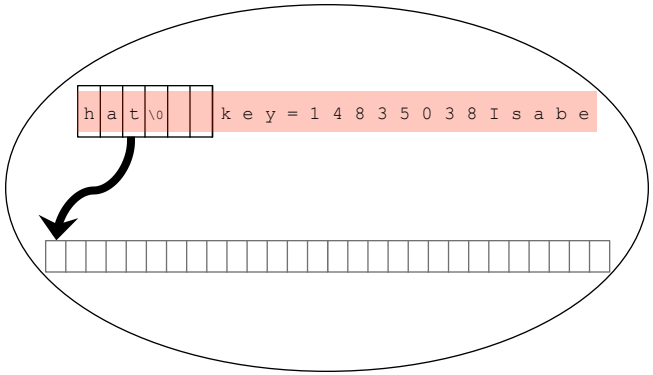
The (1) TLS and (2) DTLS implementations … do not properly handle Heartbeat Extension packets, which allows remote attackers to obtain sensitive information from process memory via crafted packets that trigger a buffer over-read, as demonstrated by reading private keys, …

Example 1: Heartbleed



from
http://xkcd.com/1354/

Example 1: Heartbleed CVE-2014-0160



Sensitive Info Uncleared Before Release

Information Exposure

Incorrect Calculation

Missing Factor

Integer Coercion

Integer Overflow Wrap-around

Incorrect Argument

Integer Underflow

Off By One

Destination Too Small

User Input Not Checked Properly

Data Too Big

No NULL Termination

Wrong Index / Pointer Out of Range

Incorrect Conversion

Buffer Overflow

Attributes:

- Access:
  - ✓*Read*, *Write*.
- Side:
  - ✓*Below* (before, under, or lower), *Above* (after, over, or upper).
- Segment (memory area):
  - ✓*Heap*, *Stack*, *BSS*, *Data* (initialized), *Code* (text)
- Method:
  - ✓*Indexed*, (bare) *Pointer*.
- Magnitude (how far outside):
  - ✓*Minimal* (just barely), *Moderate*, *Far* (e.g. 4000).
- Data Size (how much data):
  - ✓*Minimal*, *Some*, *Gazillion*.

Information Loss

Program Crash

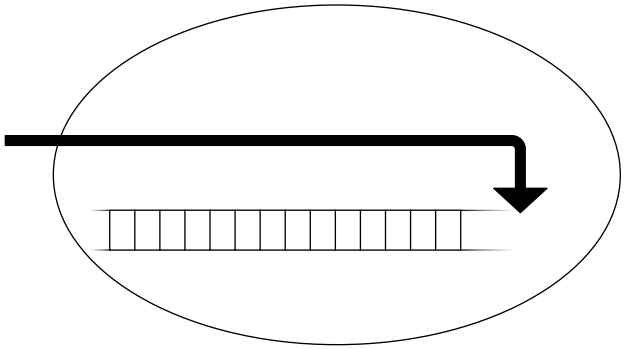
System Crash

Denial Of Service

Resource Exhaustion (Memory/CPU)

Arbitrary Code Execution

Example 2: Ghost CVE-2015-0235

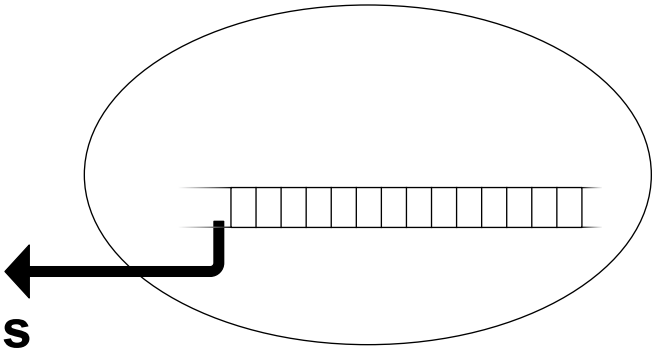


Ghost — gethostbyname buffer overflow is

- **caused by a *Destination Too Small***
- **because of an *Incorrect Calculation*, specifically *Missing Factor*,**
- **where there was a *Write* that was *After* the end by a *Moderate* number of bytes**
- **of a buffer in the *Heap***
- **that may be exploited for *Arbitrary Code Execution*.**

Heap-based buffer overflow in the __nss_hostname_digits_dots function … allows context-dependent attackers to execute arbitrary code via vectors related to the (1) gethostbyname or (2) gethostbyname2 function, aka "GHOST."

Example 3: Chrome CVE-2010-1773



Chrome WebCore — render buffer overflow is

- **caused by a *Wrong Index***
- **because of an *Incorrect Calculation*, specifically *Off by One*,**
- **where there was a *Read* that was *Below* the start by a *Minimal* amount**
- **of a buffer in the *Heap***
- **that leads to use of *User Input Not Checked Properly***
- **that may be exploited for *Information Exposure*, *Arbitrary Code Execution*, or *Program Crash* leading to *Denial of Service.***

Off-by-one error in the toAlphabetic function …, allows remote attackers to obtain sensitive information, cause a denial of service (memory corruption and application crash), or possibly execute arbitrary code via vectors related to list markers for HTML lists, …

Example 4: Refactoring CWEs

Applying our definition and attributes, Buffer Overflow CWEs can be categorized as follows.

Table 2. Buffer Overflow CWEs Organized by Attribute.

	before	after	either end	stack	heap
read	127	126	125		
write	124	120	123, 787	121	122
either r/w	786	788			