The Bugs Framework (BF) – *Your Best Friend?*

Irena Bojanova
National Institute of Standards and Technology (NIST)

NIST SATE VI, September 19, 2019

https://samate.nist.gov/BF/

Know Your Weaknesses

- They Know Your Weaknesses – Do You?

- Knowing what makes your software systems vulnerable to attacks is critical,
  → as software vulnerabilities hurt:
  security
  reliability, and
  availability of the system as a whole

- Software – should be free of ~~known~~ weaknesses (bugs)

Objective and Need

- **Objective**: Develop a complete, orthogonal, attributes based classification of software bugs that would improve dramatically on:
  - the current CWE definitions (vocabulary) used for defining software weaknesses, and
  - how vulnerability classes are described for modern software development.

- **Need**:
  - ✓ CWE is a repository of known (reported) weaknesses in the form of a nomenclature (numbered items) that has overlaps and gaps in coverage.
  - ✓ Current CWEs definitions are often *inaccurate*, *imprecise* or *ambiguous*, which makes it difficult to measure, express, and explain applicability of different software quality assurance techniques or approaches for software security.
  - ✓ Other existing classifications and guides also have their own problems related to *coverage*, *accuracy* and *precision*.

- **Gap**: Software security issues are often *described incorrectly*, and *defined inaccurately*,
  → which tremendously impacts on how threats, attacks, patches, and exposures are communicated.

BF Team and External Experts

RD:

- Irena Bojanova, NIST (Project Lead)
- Paul E. Black, NIST
- Yaacov Yesha, UMBC, NIST
- Carlos Galhardo, NIST
- Yan Wu, BGST

Website (https://samate.nist.gov/BF/):

- Farhan Nadeem
- Kyle Sung
- Zack Evans

External Collaborators:

- John Kelsy, CSD, NIST
- Rene Peralta, CSD, NIST
- Andrew Regeinscheid, CSD, NIST
- Nelson Hastings, CSD, NIST
- Kevin Greene, MITRE

→ Feel free to join us!

Resent External Presentations

- Visibility + Seeking Collaboration → Presentations To:

NITRD SPSQ IWG – July 11, 2019:

→ NSF, NASA, BLS, NIST, NOAA, NRL

NITRD CSIA IWG – August 22, 2019:

→ DISA, DHS, NSF, NRC, DARPA, NRL, ONR, OSD, DOE, DOD HPCMP, AFRL, NCO

NITRD -> NITRD GROUPS -> *SPSQ IWG*

Software Productivity, Sustainability, and Quality (SPSQ) Interagency Working Group (IWG)

NITRD -> NITRD GROUPS -> *CSIA IWG*

Cyber Security and Information Assurance (CSIA) Interagency Working Group (IWG)

Outline

1. The Bugs Framework (BF)
2. Existing Repositories of Bugs, Vulnerabilities, and Attacks
3. Problems with Current Bug Descriptions
4. Need for Structured, Precise, Orthogonal Approach
5. Developed BF Classes

The Bugs Framework (BF)

The Bugs Framework (BF)

The Bugs Framework (BF) is
a precise descriptive language for software bugs

→ allows to more *accurately* and *precisely* define software bugs and/or vulnerabilities.

← Factoring and restructuring of information in CWEs, SFPs, and STs, and classifications from NSA CAS, IDA SOAR, SEI-CERT, and more.

BF Taxonomy

BF is a set of bug classes. Each BF class:

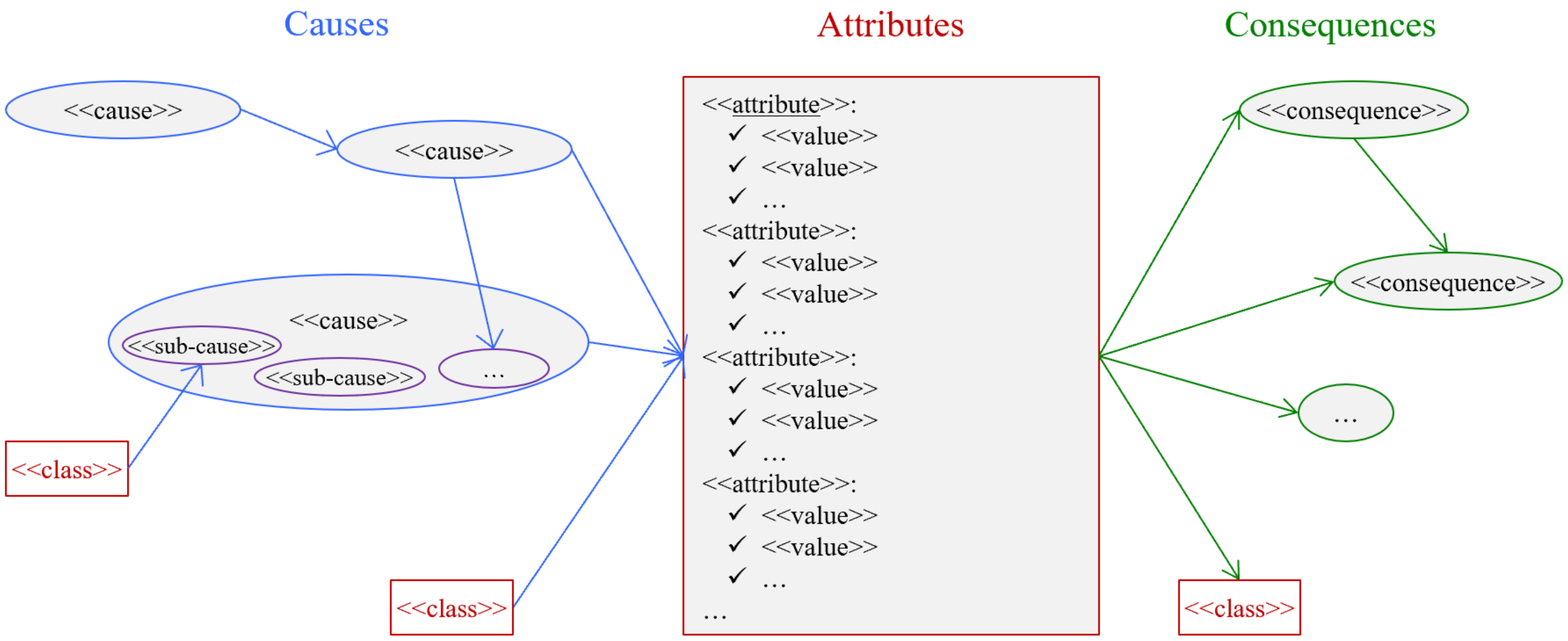
- Has an accurate and precise definition and
- Comprises:
  - ✓ Level (high or low) – identifies the fault as language-related or semantic.
  - ✓ Attributes – identify the software fault.
  - ✓ Causes – bring about the fault.
  - ✓ Consequences – to which the fault could lead.
  - ✓ Sites – locations in code where the fault might occur.

○ Sites are identifiable mainly for low level classes

→ BF uses precise definitions and terminology.

- BF is *des*criptive, not *pres*criptive.
  - ✓ It explains what happens.
  - ✓ There's not enough detail to usefully predict the result.
- BF is language independent.

BF Class Graph



- At least one attribute (underlined) identifies the software fault.
- Causes and consequences are directed graphs.

Quick Examples of BF Classes:

- **Buffer Overflow (BOF)**
- **Information Exposure (IEX)**

- **Buffer Overflow (BOF)**

BF: Buffer Overflow (BOF)

- Our Definition:

The software accesses through an array a memory location that is outside the boundaries of that array.

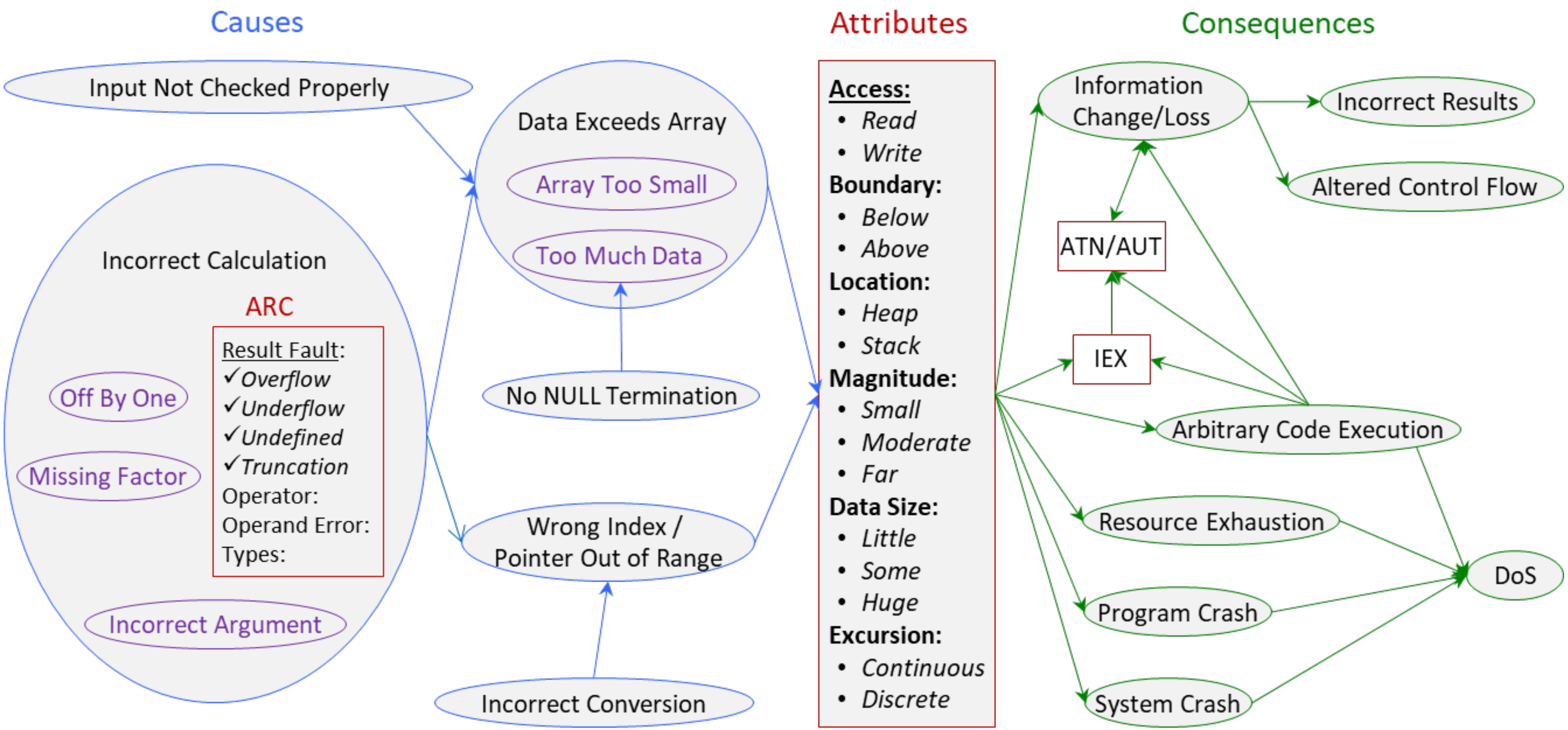
→ Clearer than CWE-119: Improper Restriction of Operations within the Bounds of a Memory Buffer: *“The software performs operations on a memory buffer, but it can read from or write to a memory location that is outside of the intended boundary of the buffer.”*

✓ clarifies that access is through the same buffer to which the intended boundary pertains.
✓ accurately, precisely, and concisely describes violation of memory safety.

Related CWEs, SFP and ST:

- CWEs are 119, 120, 121, 122, 123, 124, 125, 126, 127, 786, 787, 788, 805, 806, 823.
- SFP cluster is SFP8 Faulty Buffer Access under Primary Cluster: Memory Access.
- ST is the Buffer Overflow Semantic Template.

BOF: Causes, Attributes, and Consequences



BOF: Example – CVE-2014-0160 (Heartbleed)

CVE-2014-0160 (Heartbleed) description using BOF taxonomy:

Cause: Input Not Checked Properly leads to Data Exceeds Array (specifically, Too Much Data)
Attributes:
- Access: Read
- Boundary: Above
- Location: Heap
- Data Size: Huge
- Excursion: Continuous

Consequence: IEX (if not had been cleared)

See: https://samate.nist.gov/BF/Examples/BOF.html

BF Descriptions of BOF Related CWEs

CWE		BF Class							
ID	Name	BOF Cause(s)	BOF Attributes						BOF Consequences
			Access	Boundary	Location	Magnitude	Data Size	Excursion	
119	Improper Restriction of Operations within the Bounds of a Memory Buffer	Wrong Index/ Pointer Out of Range	any	any	any	any	any	any	any
120	Buffer Copy without Checking Size of Input ('Classic Buffer Overflow')	Array Too Small	Write	Above	any	any	any	Continuous	any
121	Stack-based Buffer Overflow	any BOF cause	Write	any	Stack	any	any	any	any
122	Heap-based Buffer Overflow	any BOF cause	Write	any	Heap	any	any	any	any
123	Write-what-where Condition	any BOF cause	Write	any	any	any	any	Discrete	any
124	Buffer Underwrite ('Buffer Underflow')	Wrong Index/ Pointer Out of Range	Write	Below	any	any	any	any	any
125	Out-of-bounds Read	PAR leads to Pointer Out of Range	Read	any	any	any	any	any	IEX
126	Buffer Over-read	any BOF cause	Read	Above	any	any	any	any	IEX
127	Buffer Under-read	Wrong Index/ Pointer Out of Range	Read	Below	any	any	any	any	IEX
786	Access of Memory Location Before Start of Buffer	Wrong Index/ Pointer Out of Range	any	Below	any	any	any	any	any
787	Out-of-bounds Write	any BOF cause	Write	any	any	any	any	any	any
788	Access of Memory Location After End of Buffer	Wrong Index/ Pointer Out of Range	any	Above	any	any	any	any	any
805	Buffer Access with Incorrect Length Value	Data Exceeds Array	any	Above	any	any	any	Continuous	any
806	Buffer Access Using Size of Source Buffer	Too Much Data (source size used)	any	any	any	any	any	Continuous	any
823	Use of Out-of-range Pointer Offset	Incorrect Calculation leads to PAR leads to Pointer Out of Range	any	any	any	any	any	Discrete	any

BOF – # of Possible Weaknesses

- Direct Causes: `(2)`
- Attributes: `(2,2,2,3,3,2)`
- Direct Consequences: (6)
- Using only the attributes Access, Boundary, Location: `8 (=2x2x2)`

- Using all the attributes: `144 (=2x2x2x3x3x2)`
  Using all the attributes, the 3 direct causes, and 6 consequences, without constraints:
  Total `2592 (= 3x(2x2x2x3x3x2)x6)`

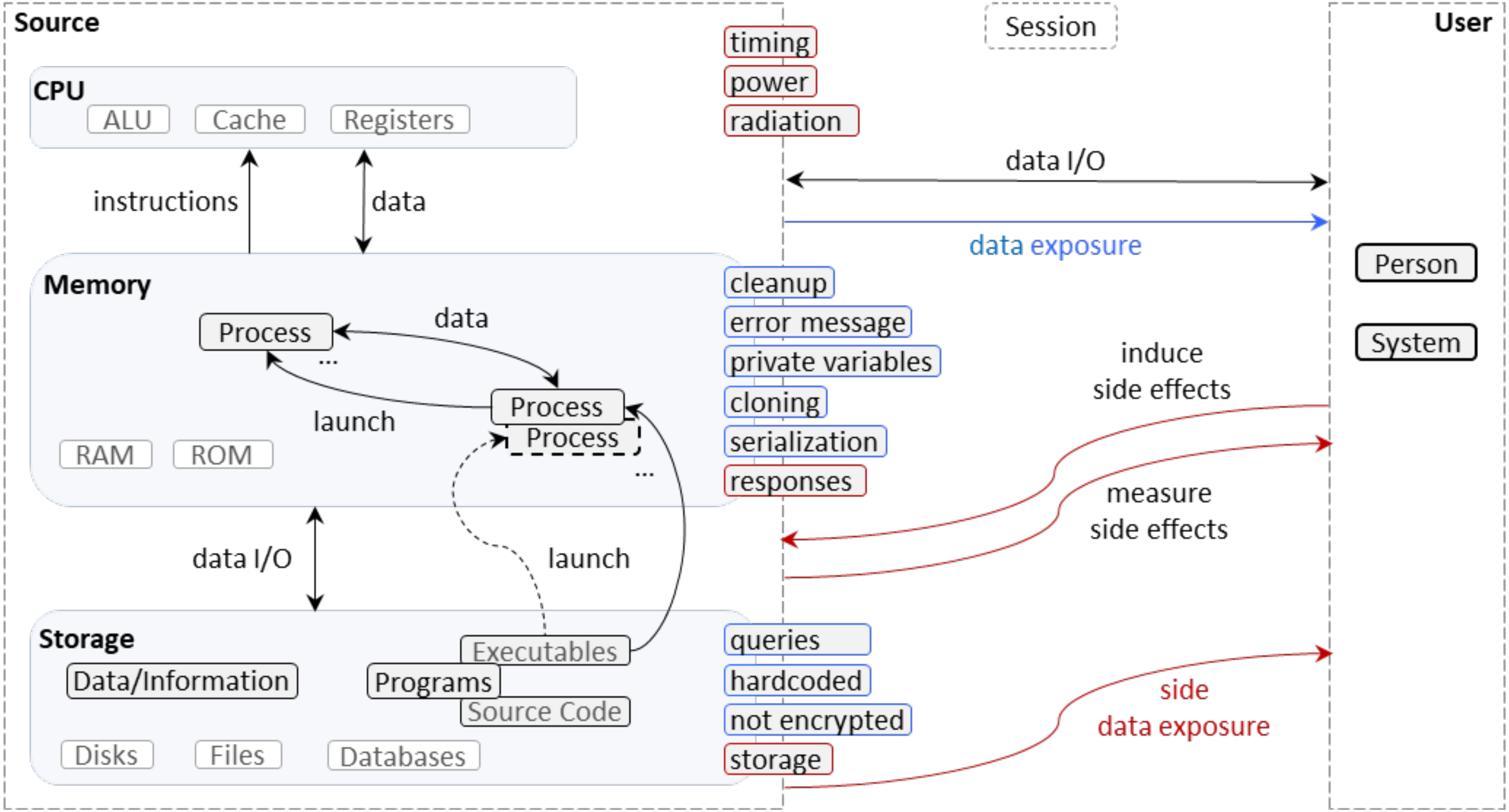
- Using all the attributes, the 3 direct causes (Array Too Small, Too Much Data, and Wrong Index / Pointer Out of Range), and the 6 direct consequences, with constraints: assuming that if the Cause is Array Too Small or Too Much Data then Boundary=Above and Excursion=Continuous.
  Total `1296 (=864+432)`

  Details:

- `864 (=6x144)` (the cases in which cause = Wrong Index / Pointer Out of Range)
- `432 (=2x(2x1x2x3x3x1)x6)` (the cases in which cause = Array Too Small or Too Much Data)

- **Information Exposure (IEX)**

BF: Information Exposure Model



BF: Information Exposure (IEX)

- Our Definition:
  Information is leaked through legitimate or side channels.

  Note that leakage to an entity that should not have information is included, not just leakage that is a security concern.

  IEX is related to: BOF, INJ, CIF, ENC, VRF, KMN, TRN, PRN.

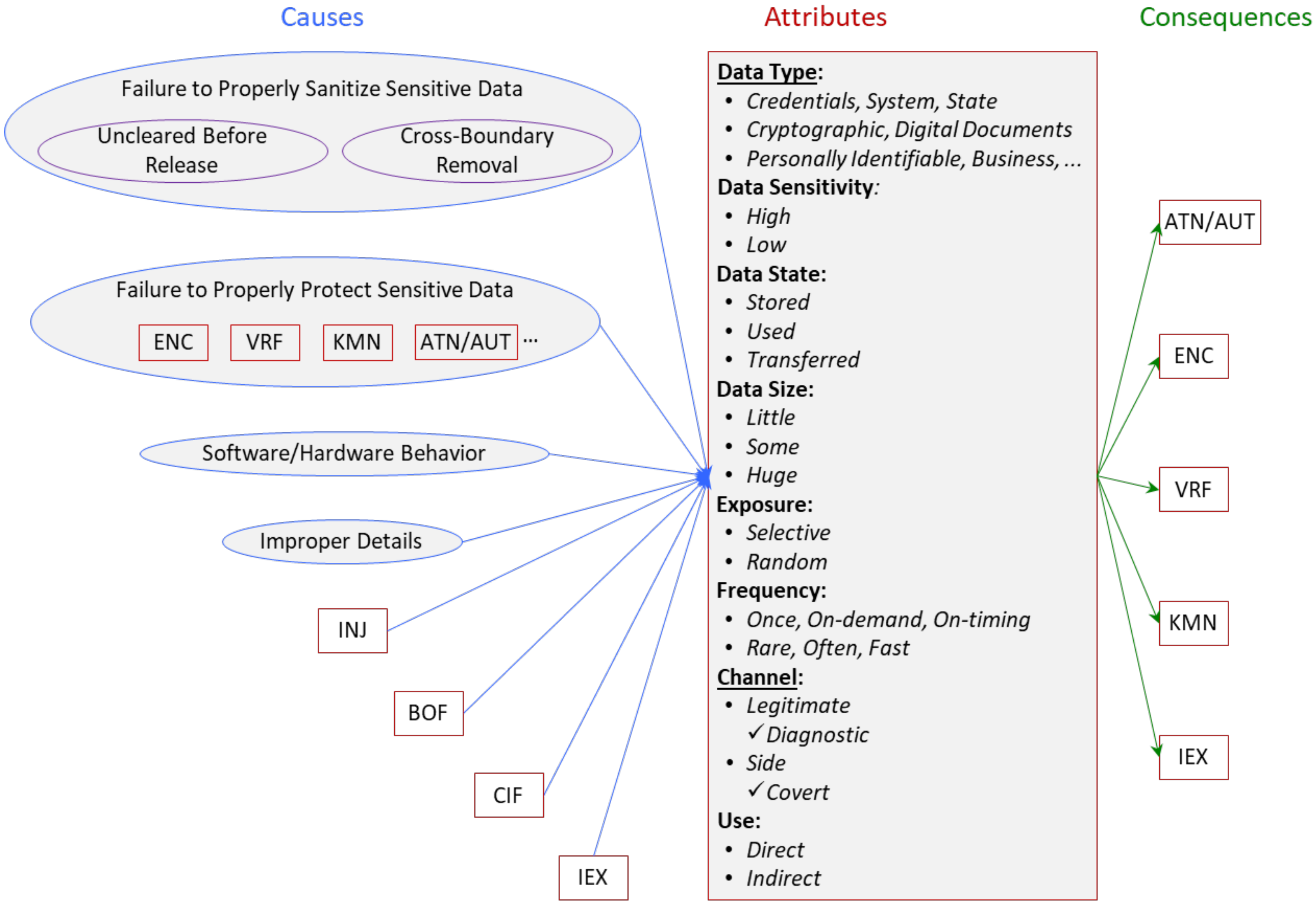
Related CWEs and SFPs:

- CWEs related to IEX are: 8, 11, 13, 200, 201, 202, 203, 204, 205, 206, 207, 208, 209, 210, 211, 212, 213, 214, 215, 226, 244, 260, 359, 377, 385, 402, 403, 433, 488, 492, 495, 497, 498, 499, 524, 514, 515, 525, 527, 528, 529, 530, 532, 535, 536, 537, 538, 539, 540, 541,546, 548, 550, 552, 555, 598, 612, 615, 642, 651, 668.

  There are many related CWEs, because information exposure can be the consequence of many weaknesses.

- The only related SFP cluster is SFP Primary Cluster: Information Leak.

IEX: Causes, Attributes, and Consequences



IEX: Example – CVE-2017-5754 (Meltdown)

CVE-2017-5754 description using IEX taxonomy:

Cause: Hardware Behavior (CPU out-of-order execution)
Attributes:
- Data Type: Any (passwords in password manager or browser, photos, emails, even business-critical documents)
- Data Sensitivity: High
- Data State: Stored (in kernel-memory registries of other processes or virtual machines in the cloud)
- Data Size: Huge
- Exposure: Selective
- Frequency: On-Demand
- Channel: Covert (cache-based timing)
- Use: Any

Consequences: Any IEX consequence.

See: https://samate.nist.gov/BF/Examples/IEX.html

BF Methodology

(Guidelines for developing and evaluation of BF classes)

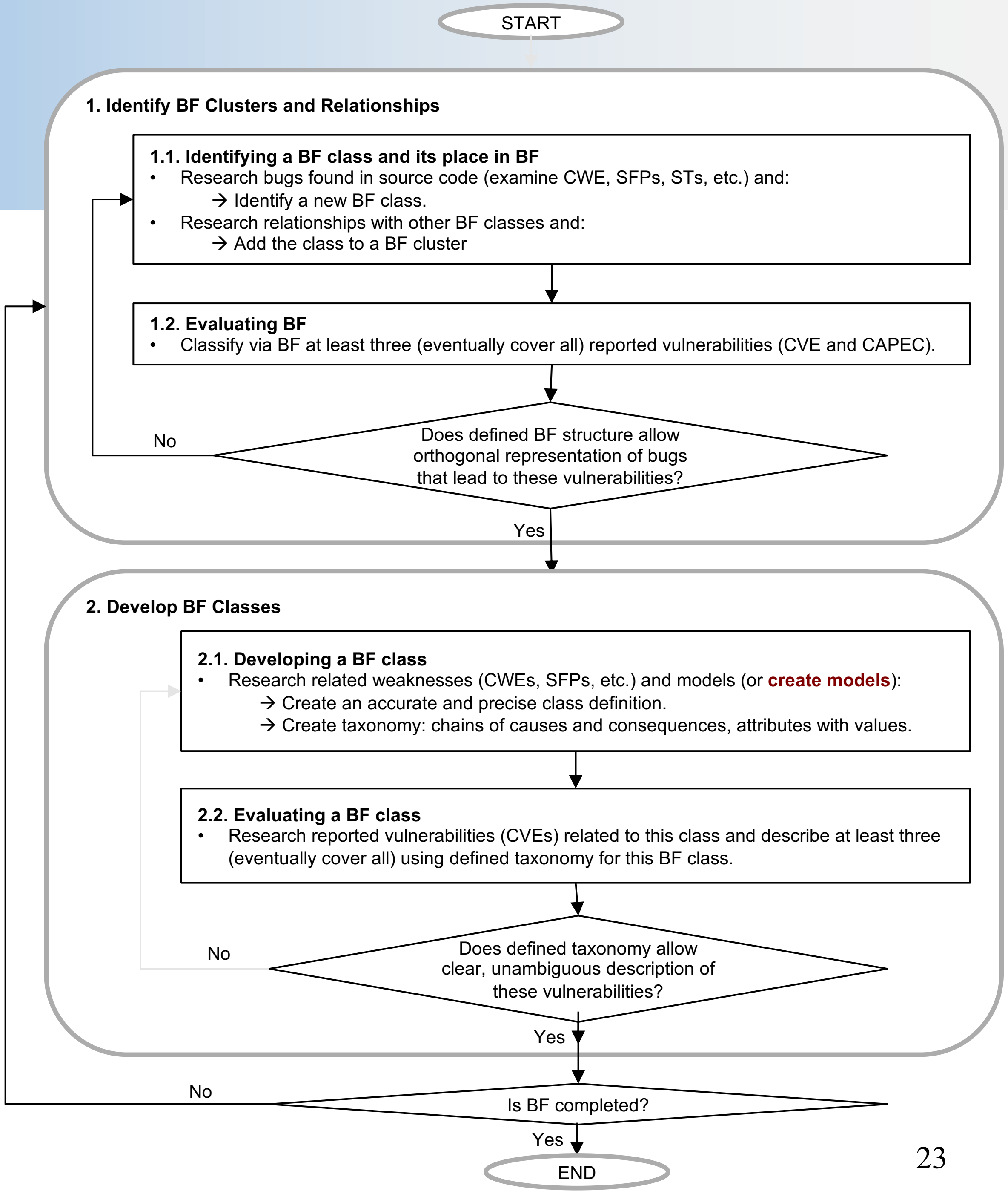
BF – complete orthogonal, attributes based classification of software bugs.

BF Class Definition:
- Concise, unambiguous description of the fault(s).
- Format: "**the software does** <<this and that wrong>>".

BF Class Taxonomy:
- **Causes**
  - ✓ What leads to the fault?
- **Consequences** (descriptive, not prescriptive)
  - ✓ What the fault leads to?
- **Attributes**
  - ✓ Focus on the failure attributes of this class.
  - ✓ What parts of the system are involved in the fault?
  - ✓ What are the details of the fault?
    - o What assumptions are violated? What parts of the definition are affected?
    - o What doesn't happen that is supposed to? What happens that is not supposed to? What exactly goes faulty (what data or resource)? How does it happen?

BF Description of a Vulnerability:
- Format: <<cause>> [(**specifically** <<sub-cause>>)] {**leads to** <<cause>> [(**specifically** <<sub-cause>>)]} [**that**] **allows** <<bug-description-via-attributes>>, **which may be exploited for** <<consequence>>{, **leading to** <<consequence>>}

  [] - "zero or one"; {} - "zero or more"



Let's Step Back for a Moment

→ **Existing Repositories of Bugs, Vulnerabilities, and Attacks**

→ **Problems?**

Repositories of Bugs, Vulnerabilities, and Attacks

- Common Weakness Enumeration (CWE)
- Software Fault Patterns (SFP)
- Semantic Templates (ST)
- NSA Center for Assured Software (CAS) Weakness Classes
- Software State-of-the-Art Resources (SOAR) Matrix
- Software Engineering Institute (SEI), Carnegie Mellon University, CERT C Coding Standard
- Common Vulnerabilities and Exposures (CVE)
- Open Web Application Security Project (OWASP): Vulnerability
- Common Attack Pattern Enumeration and Classification (CAPEC)

→ Let's take a look at them…

Common Weakness Enumeration (CWE)

CWE is a “dictionary” of **observed** bugs or flaws in software.

More than 600 distinct classes, e.g.,

- ✓ Buffer overflow
- ✓ Directory traversal
- ✓ OS injection
- ✓ Race condition
- ✓ Cross-site scripting
- ✓ Hard-coded password
- ✓ Insecure random numbers.

CWE is a community effort.

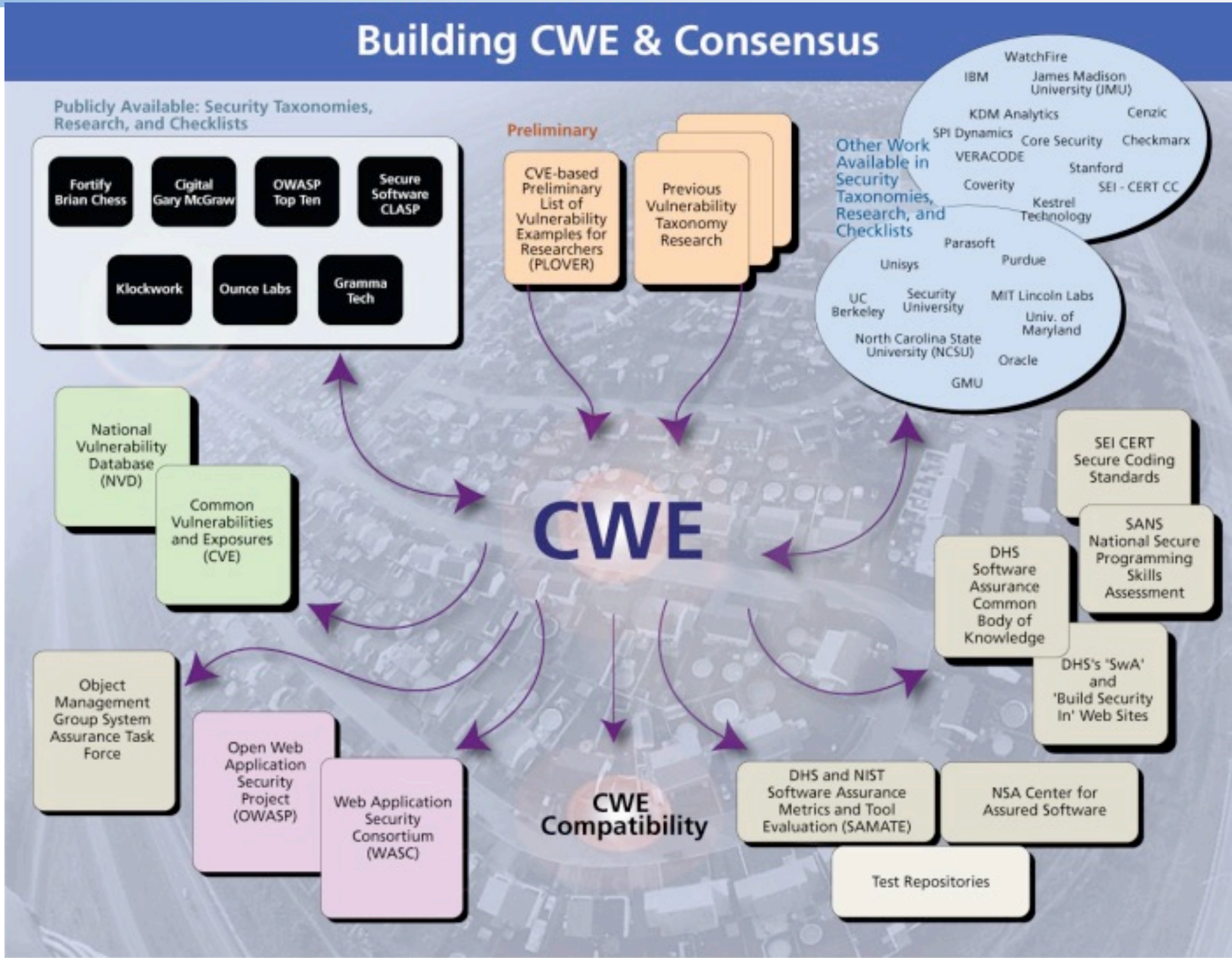


Fig. *CWE Efforts Context and Community*
[http://cwe.mitre.org/about/images/lg_consensus.jpg]

Use of CWE

CWE – for use by those who:

- Create software
- Analyze software for security flaws
- Provide tools & services for finding & defending against security flaws in software.

CWE Compatibility and Effectiveness Program:

1. CWE Searchable
2. CWE Output
3. Mapping Accuracy
4. CWE Documentation
5. CWE Coverage
6. CWE Test Results

Designations for products or services:

- ✓ CWE Compatible – meet 1) to 4)
- ✓ CWE Effective – meet all 1) to 6)

Static analysis tools:

- encouraged to map their reports to corresponding CWEs,
- so that the results from different tools could have a standard baseline to be matched and compared.

Software Fault Patterns (SFP)

- Software Fault Patterns (SFP) is a generalized description of an identifiable family of *computations* that are:
  - ✓ Described as patterns with an invariant core and variant parts
  - ✓ Aligned with injury
  - ✓ Aligned with operational views and risk through events
  - ✓ Fully identifiable in code (discernable)
  - ✓ Aligned with CWE
  - ✓ With formally defined characteristics.

→ See the clusters in Table 2 here: DoD Software Fault Patterns (go to p.26)

Software Fault Patterns (SFP)

- Software Fault Patterns (SFP): Classify, Identify patterns, Test cases generator.

- SFP are a clustering of CWEs into related weakness categories.

- Each cluster is factored into formally defined attributes, with:
  - ✓ Sites (“footholds”)
  - ✓ Conditions
  - ✓ Properties
  - ✓ Sources
  - ✓ Sinks, etc.

- SFP categories cover 632 CWEs,
- plus there are 8 deprecated CWEs.

In addition, there are:

- 21 primary clusters
- 62 secondary clusters
- 310 discernible CWEs
- 36 unique SFPs.

Semantic Templates (ST)

Semantic templates (ST) build mental models, which help us understand software weaknesses.

ST factor out chains of causes, resources and consequences that are present in CWEs.

Each ST is a human and machine understandable representation of the following phases:

1. Software faults that lead to a weakness
2. Resources that a weakness affects
3. Weakness attributes
4. Consequences/failures resulting from the weakness.

CWE-120: Buffer Copy without Checking Size of Input ('Classic Buffer Overflow')

Description Summary: The program copies an input buffer to an output buffer without verifying that the size of the input buffer is less than the size of the output buffer, leading to a buffer overflow.

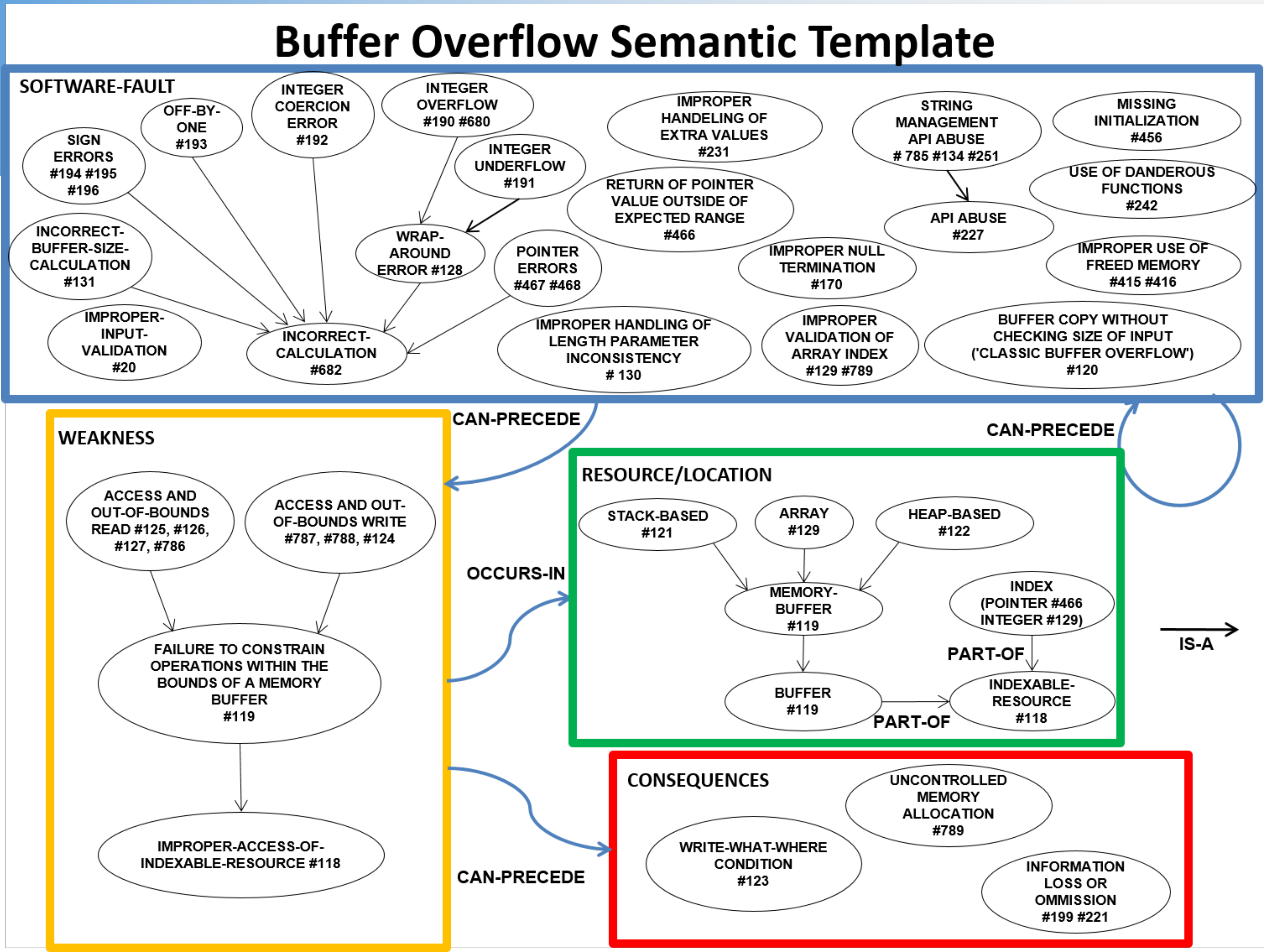
Extended Description: A buffer overflow condition exists when a program attempts to put more data in a buffer than it can hold, or when a program attempts to put data in a memory area outside of the boundaries of a buffer. The simplest type of error, and the most common cause of buffer overflows, is the "classic" case in which the program copies the buffer without restricting how much is copied.

Common Consequences: Buffer overflows often can be used to execute arbitrary code, which is usually outside the scope of a program's implicit security policy. This can often be used to subvert any other security service. Buffer overflows generally lead to crashes. Other attacks leading to lack of availability are possible, including putting the program into an infinite loop.

Fig. *Phrases in descriptions and common consequences of CWE-120, colored according to ST*: Fault, Resource/Location, Weakness, Consequence

ST

STs build mental models, which help us understand software weaknesses.



Other Repositories/Classifications

- The National Security Agency (NSA) Center for Assured Software (CAS) defines Weakness Classes in its "Static Analysis Tool Study - Methodology“
- The Software State-of-the-Art Resources (SOAR) Matrix:
  - Defines and describes a process for selecting and using appropriate analysis tools and techniques for evaluating software for software (security) assurance.
  - In particular, it identifies types of tools and techniques available for evaluating software, as well as technical objectives those tools and techniques can meet.
- Software Engineering Institute (SEI), Carnegie Mellon University, CERT C Coding Standard
- Open Web Application Security Project (OWASP): Vulnerability

→ See BF website.

Common Vulnerabilities and Exposures (CVE)
Common Attack Pattern Enumeration and Classification (CAPEC)

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

- CAPEC is a dictionary and classification taxonomy of known attacks

→ See: https://cve.mitre.org/

Problems with
Current Bug Descriptions

Problems With Current Bug Descriptions

The rise in cyberattacks lead to considerable community and government efforts to record software weaknesses, faults, failures, vulnerabilities and attacks.

→ However, *none* of the resulting repositories/enumerations are *complete* nor close to *formal*.

CWE – the Best, but also …

- CWE is widely used:
  - ✓ By far the best dictionary of software weaknesses.
  - ✓ Many tools, projects, etc. are based on CWE.

- However, in CWE:
  - ✓ For very formal, exacting work, the Definitions are often inaccurate, imprecise or ambiguous.
  - ✓ Entrees are “coarse grained” –
    each CWE bundles many stages, such as likely attacks, resources affected and consequences.
  - ✓ The coverage is uneven –
    some combinations of attributes well represented and others not appearing at all.

CWE – Imprecise Definitions

- CWE-78: Improper Neutralization of Special Elements used in an OS Command ('OS Command Injection'):

  "The software constructs all or part of an OS command using externally-influenced input from an upstream component, but it does not neutralize or incorrectly neutralizes special elements that could modify the intended OS command when it is sent to a downstream component. "

  → Note that "using input", "intended command", and "incorrectly neutralizes" are imprecise!

CWEs – Overlaps or Gaps in Coverage

e.g. Buffer Overflow

- Writes before start and after end:
  CWE-124: Buffer Underwrite (’Buffer Underflow')
  CWE-120: Buffer Copy without Checking Size of Input ('Classic Buffer Overflow')

versus

- Writes (not expressed in title) in stack and heap:
  CWE-121: Stack-based Buffer Overflow
  CWE-122: Heap-based Buffer Overflow.

- Reads before start and after end:
  CWE-127: Buffer Under-read
  CWE-126: Buffer Over-read

but

- *No reads from stack and heap.*

… while slight variants go on and on:

- 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

CWE – Imprecise Definitions

- Looking just at the cluster of buffer overflows, we see many problems.

- Here is CWE-119, the "root" of buffer overflows.

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!
→ Strictly speaking, this definition is not correct, as any variable is "a memory location that is outside of the intended boundary of the buffer."
→ Our definition says that the software can read or write *through the buffer* a memory location that is *outside* that buffer.

And, this is just one example.

CWEs – Overlaps or Gaps in Coverage

The empty cells in the table show the overlaps and gaps in the CWEs coverage of buffer overflow options with the following attributes considered:

- ✓ read/write
- ✓ before/after
- ✓ stack/heap

	Before	After	Either End		Stack	Heap
Read	CWE-127	CWE-126	CWE-125			
Write	CWE-124	CWE-120	CWE-123 CWE-787		CWE-121	CWE-122
Either R/W	CWE-786	CWE-788				

CWE – Gaps → Use of Approximate CWE

CVE-2018-19842 described with BF BOF

Cause: Boundary Not Checked Properly leads to Pointer Out or Range

Attributes:

- Access: Read
- Boundary: Above
- Location: Stack
- Data Size: Small (1 byte)
- Excursion: Continuous

Consequence: Program Crash leading to DoS

https://docs.google.com/document/d/11mbdNYAu8EH-lPsacmSYhZhOYkkiSCEDUULbYFAGkM4/edit

Note: This CVE was identified by Kevin Greene (MITRE) as an illustration of assigning an approximate CWE (specifically, the generic CWE-125 Out-of-bounds Read), because there is no exact CWE about *Read* from *Above* the boundary of a buffer on the *Stack*.

CWEs – Some are Too Generic

- E.g., CWE-118 and CWE-119 are too generic.
- They do not specifically talk about read or write, before or after, and stack or heap.
- It's like to say "Oh, this is buffer overflow. Period." and give no specifics about the particular bug you are describing.
- While it is important to be able to give the specifics about the attributes we have identified for BF BOF.

CWEs – Some are Only Causes

- E.g., CWE-680 and CWE-823 describe causes for BOF.

- While they may lead to buffer overflow bugs, these are not buffer overflow bugs themselves.

→ This is one more problem with CWE
  - ✓ some CWEs are classified not by the bug, but by a potential consequence (in this case buffer overflow) from that bug.

CWEs – Some are Too Detailed

e.g. Path Traversal – CWE for every tiny variant:

- 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: '.../...//'

Buffer overflow isn't the only cluster with problems.

Looks like, it is a waste to have CWEs
for every tiny variant of path traversal.

And if some other variant were identified,
a new CWE would have to be created.

CWEs – Not Always Easy to Find

- Example: How to figure out this CWE is related to Information Exposure

  CWE-433: Unparsed Raw Web Content Delivery

- SFP researchers found it by an automated process and put it in
  SFP Secondary Cluster: Exposed Data.

- But if person had to do this, search in CWE does not help much.

Software Fault Patterns (SFP) – Improve on CWEs

- SFP overcomes the problem of combinations of attributes in CWE.

→ For example, the SFP factored attributes are more clear than the irregular coverage of CWEs.

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

Summary: 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.

Extended description: Certain languages allow direct addressing of memory locations and do not automatically ensure that these locations are valid for the memory buffer that is being referenced. This can cause read or write operations to be performed on memory locations that may be associated with other variables, data structures, or internal program data. As a result, an attacker may be able to execute arbitrary code, alter the intended control flow, read sensitive information, or cause the system to crash.

CWE-120: Buffer Copy without Checking Size of Input ('Classic Buffer Overflow')

Summary: The program copies an input buffer to an output buffer without verifying that the size of the input buffer is less than the size of the output buffer, leading to a buffer overflow.

Extended Description: A buffer overflow condition exists when a program attempts to put more data in a buffer than it can hold, or when a program attempts to put data in a memory area outside of the boundaries of a buffer.

Common Consequences: Buffer overflows often can be used to execute arbitrary code. Buffer overflows generally lead to crashes.

Parameters	Buffer location		Access kind		Access position		Boundary exceeded	
	heap	stack	write	read	inside	outside	lower	upper
119 - Improper Restriction of Operations within Bounds of Buffer	√	√	√	√		√	√	√
120 - Buffer Copy without Checking Size of Input	√	√	√		√		√	√
121 - Stack Overflow		√	√		√		√	√
122 - Heap Overflow	√		√		√		√	√
123 - Write-what-where Condition	√	√	√				√	√
124 - Buffer Underwrite	√	√	√			√	√	
125 - Out-of-bounds read	√	√		√			√	√
126 - Buffer Overread	√	√		√		√		√
127 - Buffer Underread	√	√		√		√	√	

Semantic Templates (ST) – Improve on CWEs, too

- STs build mental models, which help us understand software weaknesses.
- Each ST is a human and machine understandable representation of:
  1. Software faults that lead to a weakness
  2. Resources that a weakness affects
  3. Weakness attributes
  4. Consequences/failures resulting from the weakness.

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

Summary: 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***.

Extended description: Certain languages allow direct addressing of ***memory locations*** and ***do not automatically ensure that these locations are valid for the memory buffer that is being referenced***. This can ***cause read or write operations to be performed on memory locations that may be associated with other variables, data structures, or internal program data***. As a result, an attacker may be able to ***execute arbitrary code***, ***alter the intended control flow***, ***read sensitive information***, or ***cause the system to crash***.

CWE-120: ***Buffer Copy without Checking Size of Input*** ('Classic ***Buffer Overflow***')

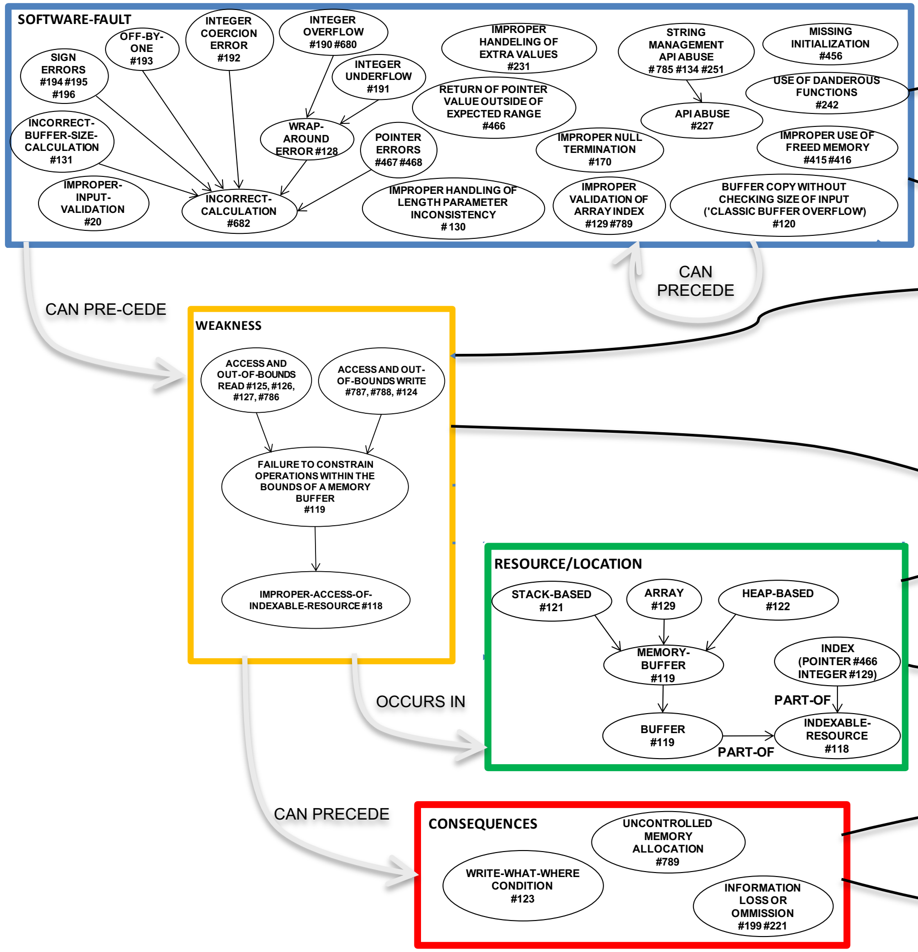
Summary: The program copies an input ***buffer*** to an output ***buffer*** ***without verifying that the size of the input buffer is less than the size of the output buffer***, leading to a ***buffer overflow***.

Extended Description: A ***buffer overflow*** condition exists when a ***program attempts to put more data in a buffer than it can hold, or when a program attempts to put data in a memory area outside of the boundaries of a buffer***.

Common Consequences: ***Buffer overflows*** often can be used to ***execute arbitrary code***. ***Buffer overflows*** generally ***lead to crashes***.

Parameters	Buffer location		Access kind		Access position		Boundary exceeded	
	heap	stack	write	read	inside	outside	lower	upper
119 - Improper Restriction of Operations within Bounds of Buffer	√	√	√	√		√	√	√
120 - Buffer Copy without Checking Size of Input	√	√	√		√		√	√
121 - Stack Overflow		√	√		√		√	√
122 - Heap Overflow	√		√		√		√	√
123 - Write-what-where Condition	√	√	√				√	√
124 - Buffer Underwrite	√	√	√			√	√	
125 - Out-of-bounds read	√	√		√			√	√
126 - Buffer Overread	√	√		√		√		√
127 - Buffer Underread	√	√		√		√	√	

Semantic Templates (STs) – Improve on CWEs, too



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

Summary: 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*.

Extended description: Certain languages allow direct addressing of *memory locations* and *do not automatically ensure that these locations are valid for the memory buffer that is being referenced*. This can *cause read or write operations to be performed on memory locations that may be associated with other variables, data structures, or internal program data*. As a result, an attacker may be able to *execute arbitrary code*, *alter the intended control flow*, *read sensitive information*, or *cause the system to crash*.

CWE-120: *Buffer Copy without Checking Size of Input* ('Classic *Buffer Overflow*')

Summary: The program copies an input *buffer* to an output *buffer without verifying that the size of the input buffer is less than the size of the output buffer*, leading to a *buffer overflow*.

Extended Description: A *buffer overflow* condition exists when a *program attempts to put more data in a buffer than it can hold, or when a program attempts to put data in a memory area outside of the boundaries of a buffer*.

Common Consequences: *Buffer overflows* often can be used to *execute arbitrary code*. *Buffer overflows* generally *lead to crashes*.

Parameters	Buffer location		Access kind		Access position		Boundary exceeded	
	heap	stack	write	read	inside	outside	lower	upper
119 - Improper Restriction of Operations within Bounds of Buffer	√	√	√	√		√	√	√
120 - Buffer Copy without Checking Size of Input	√	√	√		√		√	√
121 - Stack Overflow		√	√		√		√	√
122 - Heap Overflow	√		√		√		√	√
123 - Write-what-where Condition	√	√	√				√	√
124 - Buffer Underwrite	√	√	√			√	√	
125 - Out-of-bounds read	√	√		√			√	√
126 - Buffer Overread	√	√		√		√		√
127 - Buffer Underread	√	√		√		√	√	

But SFP & ST Also Have Problems

- Software Fault Patterns (SFP):
  - ✓ are an excellent advance
  - ✓ "factor" weaknesses into parameters,
  - ✓ But:
    - do not include upstream causes or consequences, and
    - are based solely on CWEs.

→ SFPs do not tie fault clusters to:
  - causes or chains of fault patterns
  - consequences of a particular vulnerability.

→ Since SFP were derived from CWEs, more work is needed for embedded or mobile concerns, such as, battery drain, physical sensors (e.g. Global Positioning System (GPS) location, gyroscope, microphone, camera) and wireless communications.

Note: SFP is coupled with a meta-language, Semantics of Business Vocabularies and Rules (SBVR), in which causes, threats, consequences, etc. may be expressed. However, SFP does not have an integrated means of expressing them.

But SFP & ST Also Have Problems

- Semantic Templates (ST):
  - ✓ Collect CWEs into four general areas:
    - Software-fault
    - Weakness
    - Resource/Location
    - Consequences.
  - ✓ But:
    - are only guides to aid human comprehension.

- The other existing bug descriptions also have their own limitations.
- They are based on CWEs and don't go beyond CWEs.

➔ Need for Structured, Precise, Orthogonal Approach

Need for Structured, Precise, Orthogonal Approach

- Without accurate and precise classification and comprehension of all possible types of software bugs, the development of reliable software will remain extremely challenging.

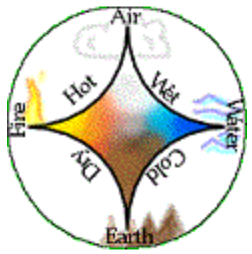
- As a result the newly delivered and the legacy systems will continue having security holes despite all the patching to correct errant behavior.

We don’t (yet) know the best structure for bugs descriptions.

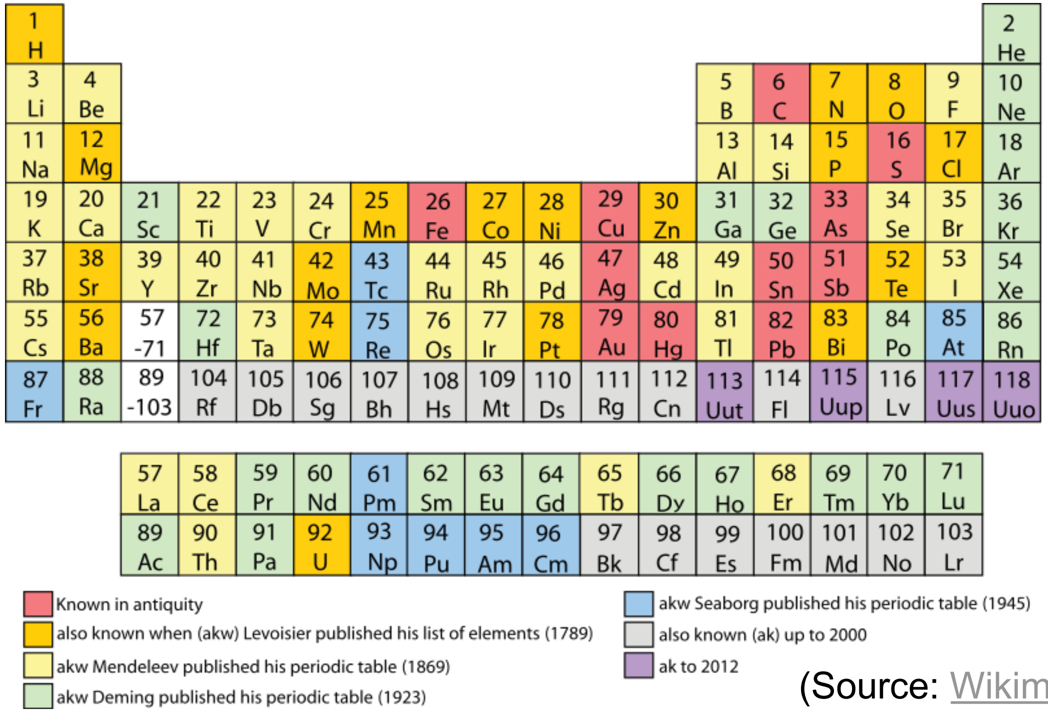
But, for analogies on what we are embarking on, let’s look at
some well-know organizational structures in science …

Periodic Table & Others to Describe Molecules

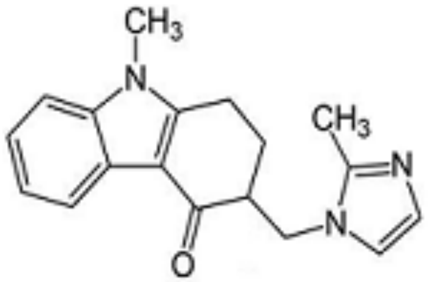
- Greeks used the terms element and atom.
  Aristotle: substances are a mix of Earth, Fire, Air, or Water.
- Alchemists cataloged substances, such as alcohol, sulfur, mercury, and salt.
  (note: Lavoisier had light and caloric on his 33 elements list!)
- Periodic table reflects atomic structure & forecasts properties of missing elements.



(Source: Reich Chemistry)



(Source: Wikimedia Commons)



$C_{18}H_{19}N_3O$

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

Zofran ODT has a chemical formula ($C_{18}H_{19}N_3O$), structural formula (picture), and a detailed name.

Tree of Life

Discoveries of more than 1,000 new types of Bacteria and Archaea over the past 15 years have dramatically rejiggered the Tree of Life to account for these microscopic life forms.

- Divides life into three domains:
  - ✓ Bacteria
  - ✓ Archaea
  - ✓ Eukaryotes.
- Clearly shows "life we see around us – plants, animals, humans" and other Eukaryotes – represent a tiny percentage of world's biodiversity.

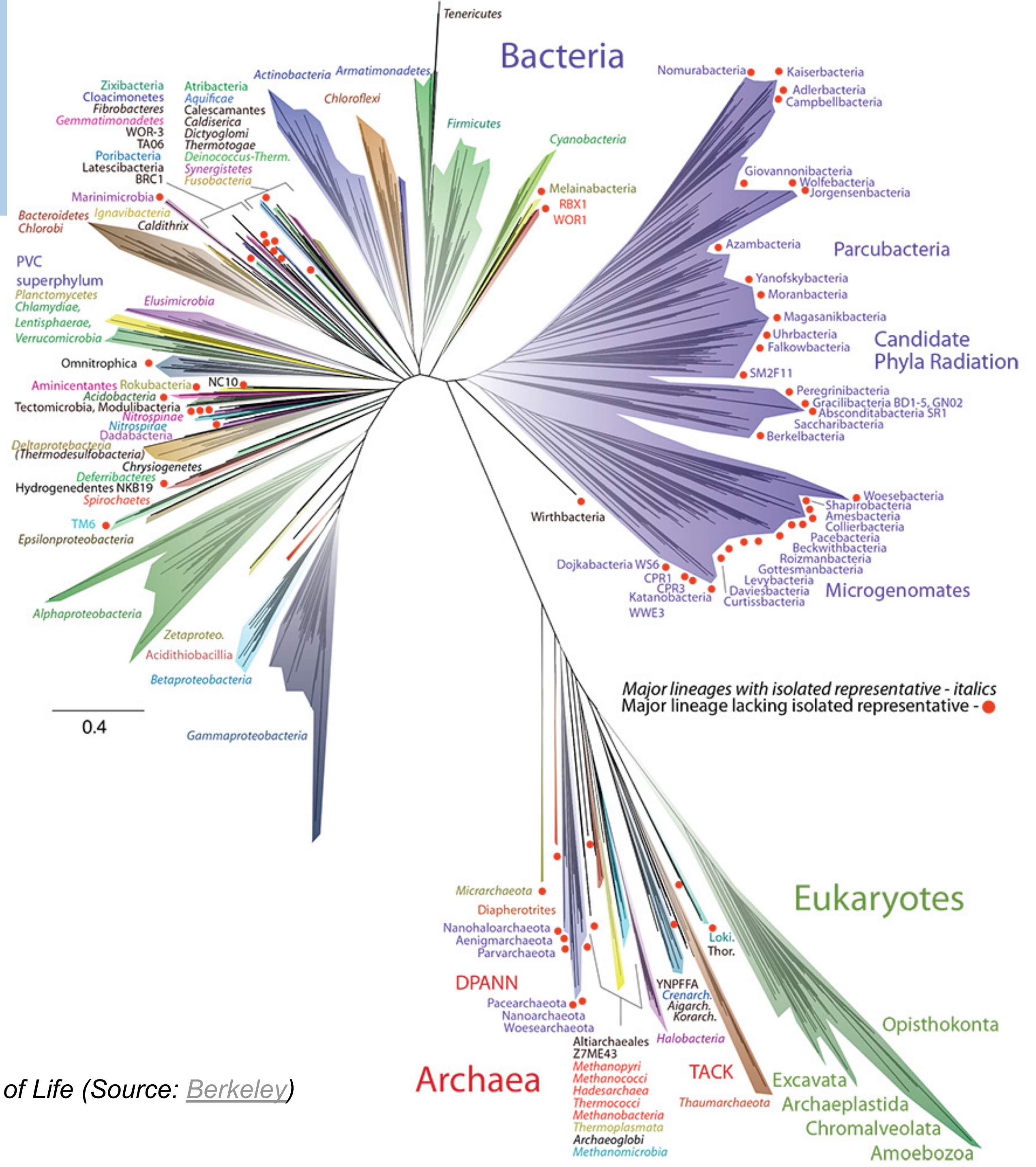
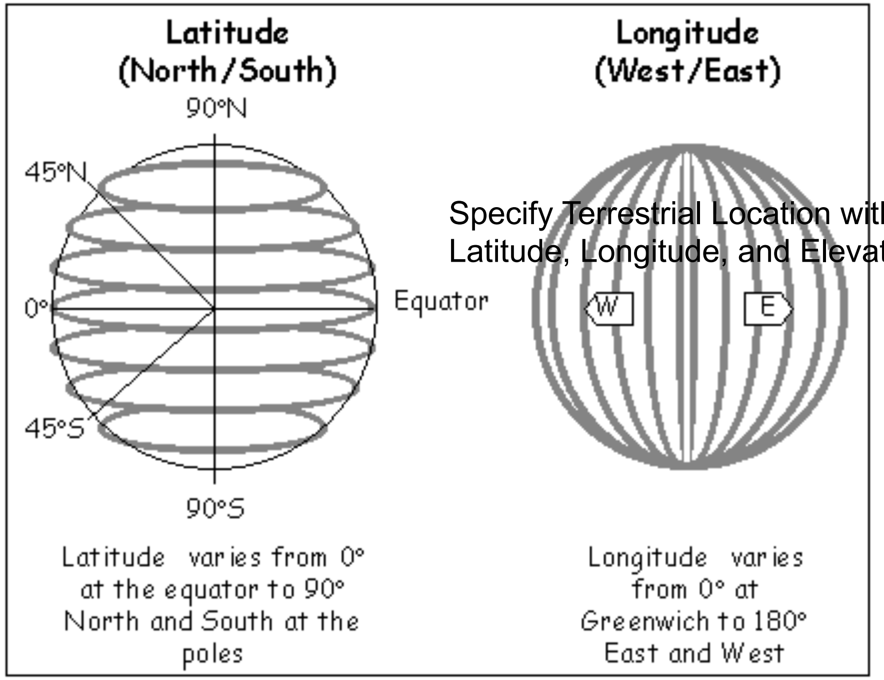


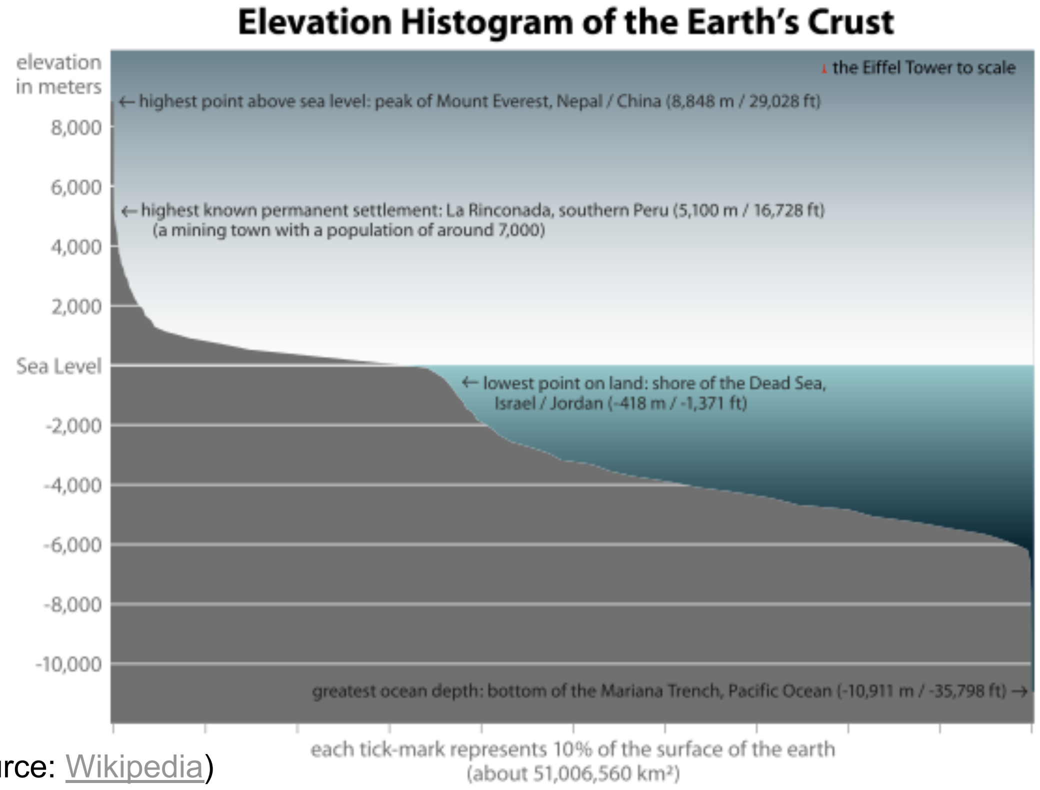
Fig. *The Tree of Life (Source: Berkeley)*

Geographic Coordinate System

Specify Any Terrestrial Location using Latitude, Longitude, and Elevation.



Specify Terrestrial Location with
Latitude, Longitude, and Elevation



Geographic Coordinate System (Source: Wikipedia)

Precise Medical Language

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

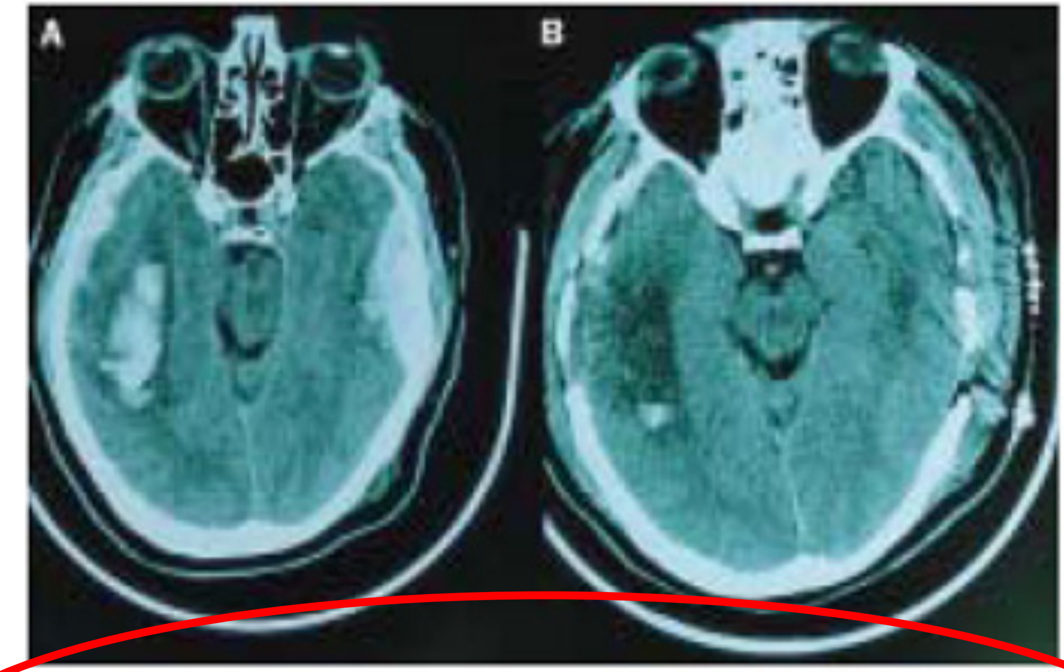


Figure 2: **Computed tomography of a comatose patient with a left temporal epidural haematoma, right parenchymal temporal lobe haematoma, and a right convexity subdural haematoma before and after craniotomy and evacuation of haematomas**

- The caption uses precise medical terminology.
- They are not trying to obfuscate.
- They are "painting a picture" (adding arrows and circles) with words.

→ So, just as a doctor would be hampered by only being able to say, “this thingy here”, software assurance work is more difficult, because of the lack of a precise common vocabulary (ontology).

(Source: http://i.stack.imgur.com/uLH9P.jpg)

Current BF Classes

Current BF Classes

1. Information Exposure (IEX)
2. Randomness Cluster (RND)
   - Pseudo-Random Number Bugs (PRN)
   - True-Random Number Bugs (TRN)
3. Cryptography Cluster (CRY)
   - Key Management Bugs (KMN)
   - Encryption Bugs (ENC)
   - Verification Bugs (VRF)
4. Access Control Cluster (ACC)
   - Identity Proofing (IDP)
   - Authentication Bugs (ATN)
   - Authorization Bugs (ATZ)
5. Memory Cluster (MEM)
   - Memory Allocation Bugs (MAL)
   - Memory Use Bugs (MUS)

     Buffer Overflow (BOF) → refined
6. Injection (INJ) → refined
7. Control of Interaction Frequency (CIF) → refined

Benefits of Using BF

Benefits of Using BF

BF provides a superior, unified approach that allows us to:

- Precisely and unambiguously express software bugs or vulnerabilities.
- Explain clearly applicability and utility of different software quality or assurance techniques or approaches.
- More formally reason about assurance techniques or mitigation approaches that may work for a fault with certain attributes (but not for the same fault with other attributes).

Benefits of Using BF

With BF practitioners and researchers can more accurately, precisely and clearly:

- Describe problems in software.
- Clearly document the classes of bugs that a tool does and does not report.
- Explain what vulnerabilities the proposed techniques prevent.

- Those concerned with software quality, reliability of programs and digital systems, or cybersecurity
  → will be able to make more rapid progress by more clearly labeling the results of errors in software.
- Those responsible for designing, operating and maintaining computer complexes
  → can communicate with more exactness about threats, attacks, patches and exposures.

BF: Future Work

- One of our next steps is to explain more vulnerabilities using the developed BF classes.
- Another step is to develop more and more BF classes.

→ Our goal is for BF to become the software developers’ and testers’ “*Best Friend*.”

 62

Questions

https://samate.nist.gov/BF/