The Bugs Framework (BF)

Software developer's and tester's "Best Friend"

Irena Bojanova Paul E. Black Yaacov Yesha

National Institute of Standards and Technology (NIST)

Yan Wu

Bowling Green State University (BGSU)

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

NIST National Institute of Standards and Technology • U.S. Department of Commerce

Context

- Many federal regulations, guidelines and policies call for:
  - ✓ Security assurance and
  - ✓ Acquisition/use of tested and evaluated software.

- For that a formal methodology for precisely and unambiguously expressing all types of software weaknesses (bugs) and thorough tools for systematically assuring that software does not contain such bugs are desperately needed.

- The government and US industries are facing significant uncertainty about software security, reliability and availability as a result of the current state-of-the-art.

Definitions

- Software Weakness (Bug): "A piece of code that may lead to a vulnerability." [1]
- Security Vulnerability: "A property of system requirements, design, implementation, or operation that could be accidentally triggered or intentionally exploited and result in a security failure." [1]
- Software Attack: "The use of an exploit(s) by an adversary to take advantage of a weakness(s) with the intent of achieving a negative technical impact(s). An attack includes the entire "Cyber Attack Lifecycle" reconnaissance, weaponize, deliver, exploit, control, execute, and maintain." [2]
- Security Failure: "Any event that is a violation of a particular system's explicit or implicit security policy." [1]
  - ✓ "the source of any failure is a latent vulnerability." [1]
  - ✓ "if there is a failure, there must have been a vulnerability." [1]
- Source Code: "A series of statements written in a human-readable computer programming language." [1]

Note: A vulnerability is the result [of the exploitation] of one or more weaknesses in requirements, design, implementation, or operation. Sometimes a weakness can never result in a failure, in which case it is not exploitable and not a vulnerability. Such a weakness might be masked by another part of the software or might only cause a failure in combination with another weakness. Thus we use the term "weakness" instead of "flaw" or "defect." [1]

References

[1] Black, P., Kass, M., Koo. M., Fong, E. Source Code Security Analysis Tool Functional Specification Version 1.1, NIST Special Publication 500-268 v1.1.
[2] The MITRE Corporation. Common Attack Pattern Enumeration and Classification (CAPEC), Glosary, Attack.

Questions

- What is the entire hierarchy of orthogonal (non-overlapping) classes of software bugs?
- What is the accurate and precise definition of each of these classes?
- Are there different kinds of relationships between these classes (e.g., are authentication bugs in an "is-a" relationship with information exposure bugs and/or cryptography related bugs)?
- Which are the exclusively mobile specific classes of bugs?
- What are the characteristics/attributes of each bug class that would allow to precisely describe any related software vulnerabilities?
- What are the chains of causes and the possible consequences of a bug in software? These should be a finite number and should be defined precisely.

Goal

→ Create an unambiguous framework for expressing software bugs.

- Currently there are no government, industry or international standards as to what constitutes the complete hierarchical classification of software bugs or a unified, formal approach for unambiguously expressing software vulnerabilities and chains of failures.
- Software developers, testers, and researchers need ways to:
  - ✓ Accurately and precisely comprehend and avoid all possible types of software bugs.
  - ✓ Develop techniques for making repeatable, quantified measurements of software quality and assurance.
  - ✓ Explain clearly applicability and utility of different software quality and assurance techniques or approaches.
  - ✓ Formally reason about assurance techniques or mitigation approaches that may work for a fault with certain attributes, but not for the same general kind of fault that has other attributes.
  - ✓ Estimate risk and determine best mitigation strategies based on known consequences of different kinds of faults.

Outline

- Problems With Current Bug Descriptions
- Need for Structured Approach
- The Bugs Framework (BF)
- Examples of Applying BF
- Next Steps
- Benefits

Common Nomenclature

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

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
  - ✓ insecure random numbers.

http://cwe.mitre.org/

Common Vulnerability Enumeration (CVE)

- 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.

https://cve.mitre.org/

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

CWE – the Best, but also a Mess

- CWE is widely used:
  - ✓ By far the best dictionary of software weaknesses.
  - ✓ Many tools, projects, etc. are based on CWE.
- However, in CWE:
  - ✓ Definitions are imprecise and inconsistent.
  - ✓ Entrees are "coarse grained" –
    bundle lots of stuff, like consequences and likely attacks.
  - ✓ The coverage is uneven:
    some combinations well represented and others not represented at all.
  - ✓ No mobile weaknesses, e.g., battery drain, physical sensors (GPS, gyro, microphone, hi-res camera), unencrypted wireless communication, etc.

CWE – Imprecise Definitions

- 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!

- 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 – 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

CWEs – 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: '.../...//'

Software Fault Patterns (SFP) – Improve on CWEs

- 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 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.

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

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

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

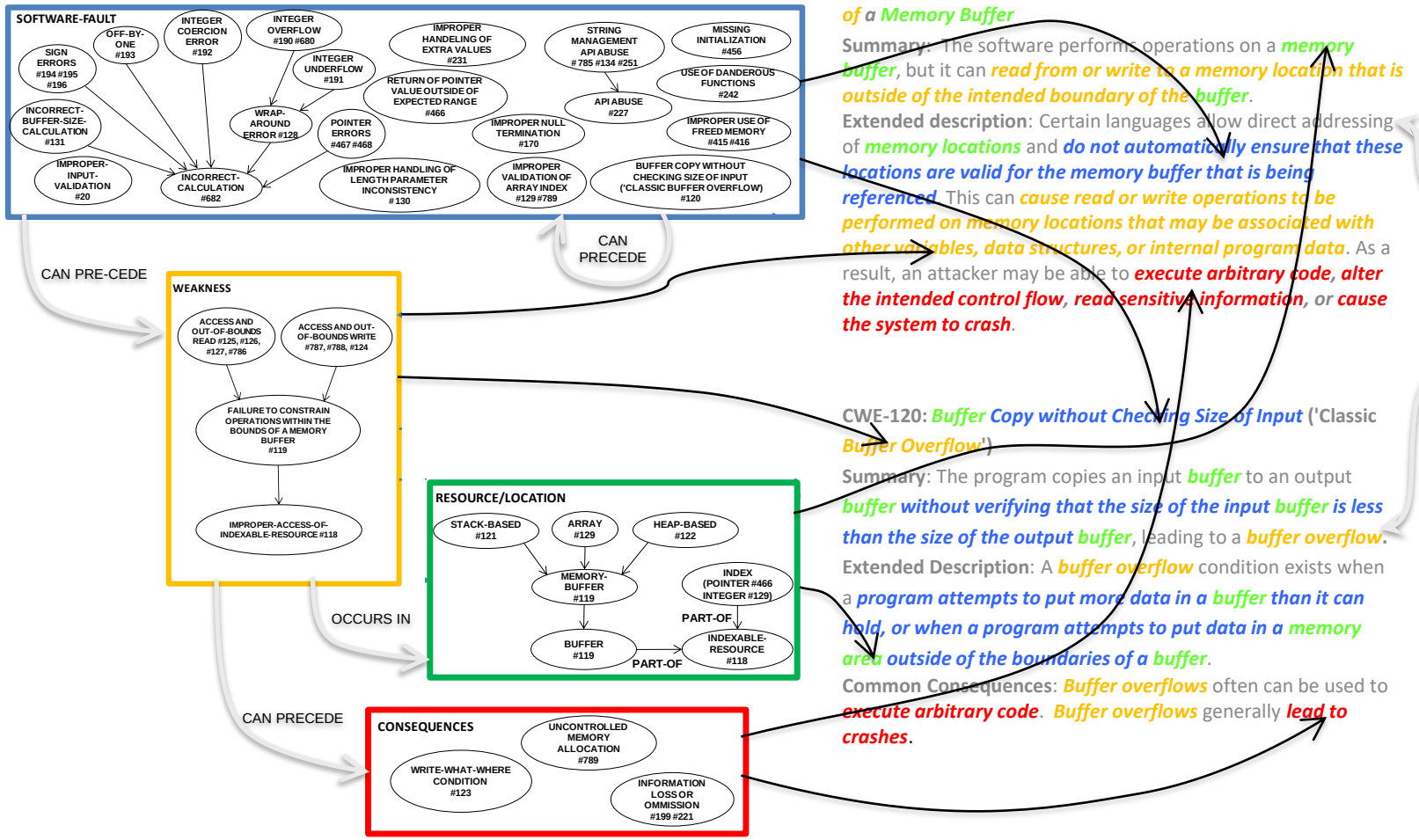
Summary: The software performs operations on a *memory buffer*, but it can *read from or write to a memory location that is outside of the intended boundary of the buffer*.
Extended description: Certain languages allow direct addressing of *memory locations* and *do not automatically ensure that these 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



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

Outline

- Problems With Current Bug Descriptions
- Need for Structured Approach
- The Bugs Framework (BF)
- Examples of Applying BF
- Next Steps
- Benefits

Need of Structured Approach

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

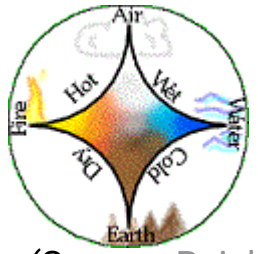
- As a result the newly delivered and the legacy systems will continue having security holes despite all the patching to correct errant behavior.

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.



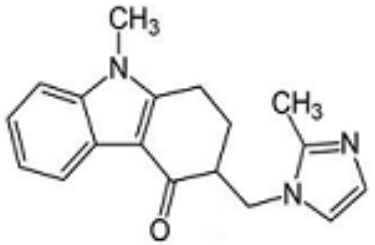
(Source: Reich Chemistry)

1 H																	2 He
3 Li	4 Be											5 B	6 C	7 N	8 O	9 F	10 Ne
11 Na	12 Mg											13 Al	14 Si	15 P	16 S	17 Cl	18 Ar
19 K	20 Ca	21 Sc	22 Ti	23 V	24 Cr	25 Mn	26 Fe	27 Co	28 Ni	29 Cu	30 Zn	31 Ga	32 Ge	33 As	34 Se	35 Br	36 Kr
37 Rb	38 Sr	39 Y	40 Zr	41 Nb	42 Mo	43 Tc	44 Ru	45 Rh	46 Pd	47 Ag	48 Cd	49 In	50 Sn	51 Sb	52 Te	53 I	54 Xe
55 Cs	56 Ba	57 -71	72 Hf	73 Ta	74 W	75 Re	76 Os	77 Ir	78 Pt	79 Au	80 Hg	81 Tl	82 Pb	83 Bi	84 Po	85 At	86 Rn
87 Fr	88 Ra	89 -103	104 Rf	105 Db	106 Sg	107 Bh	108 Hs	109 Mt	110 Ds	111 Rg	112 Cn	113 Uut	114 Fl	115 Uup	116 Lv	117 Uus	118 Uuo

57 La	58 Ce	59 Pr	60 Nd	61 Pm	62 Sm	63 Eu	64 Gd	65 Tb	66 Dy	67 Ho	68 Er	69 Tm	70 Yb	71 Lu
89 Ac	90 Th	91 Pa	92 U	93 Np	94 Pu	95 Am	96 Cm	97 Bk	98 Cf	99 Es	100 Fm	101 Md	102 No	103 Lr

Known in antiquity
also known when (akw) Levoisier published his list of elements (1789)
akw Mendeleev published his periodic table (1869)
akw Deming published his periodic table (1923)
akw Seaborg published his periodic table (1945)
also known (ak) up to 2000
ak to 2012

(Source: Wikimedia Commons)



$C_{18}H_{19}N_3O$

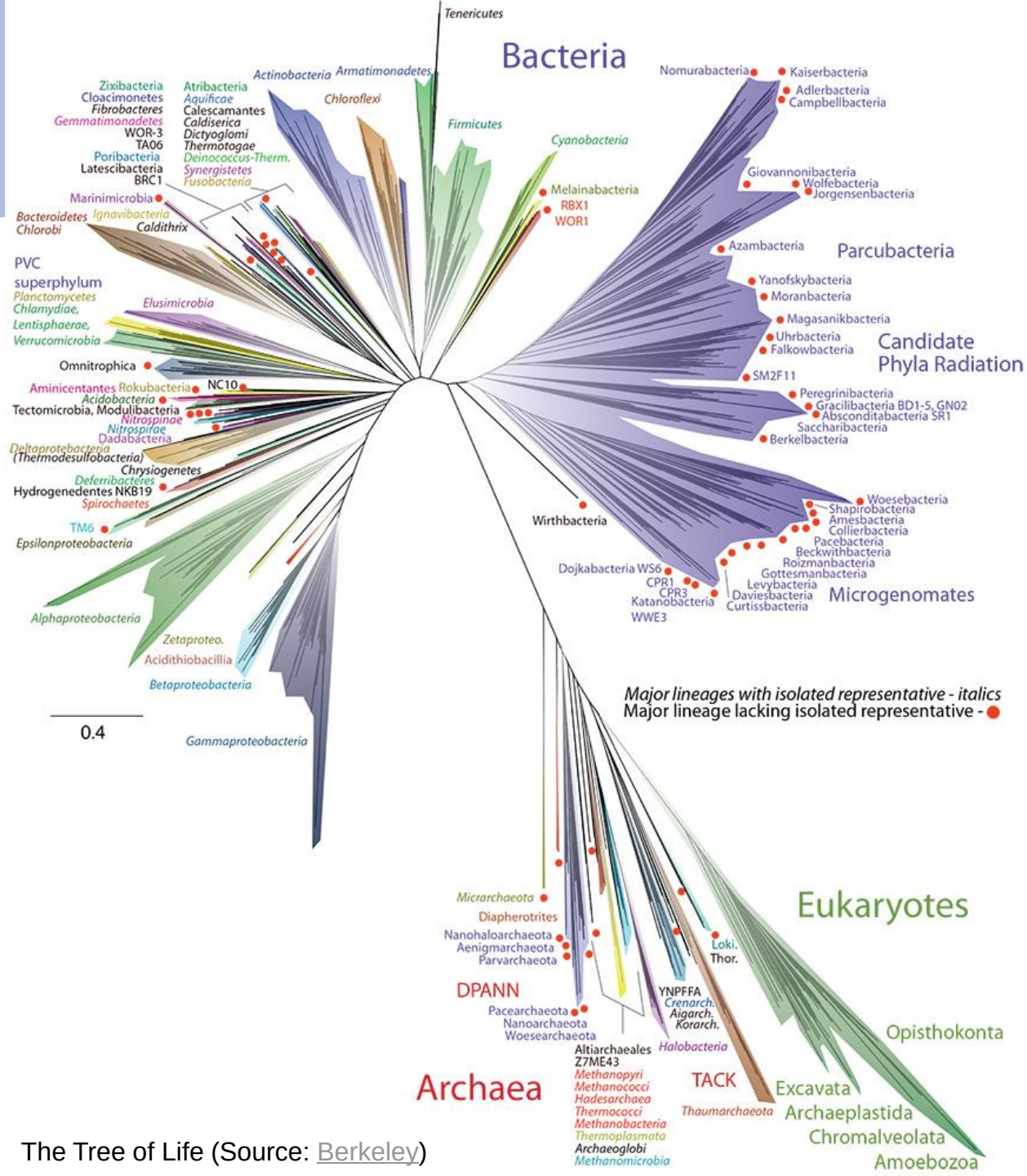
(±) 1, 2, 3, 9-tetrahydro-9-methyl-3-[(2-methyl-1H-imidazol-1-yl)methyl]-4H-carbazol-4-one

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.

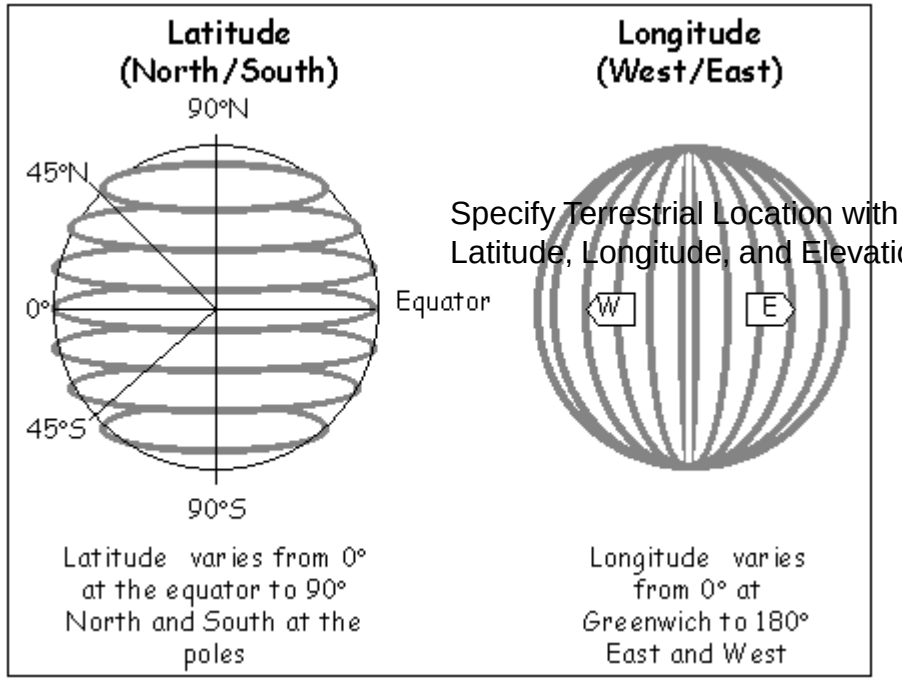
- Clearly shows "life we see around us – plants, animals, humans" and other Eukaryotes – represent a tiny percentage of world's biodiversity.



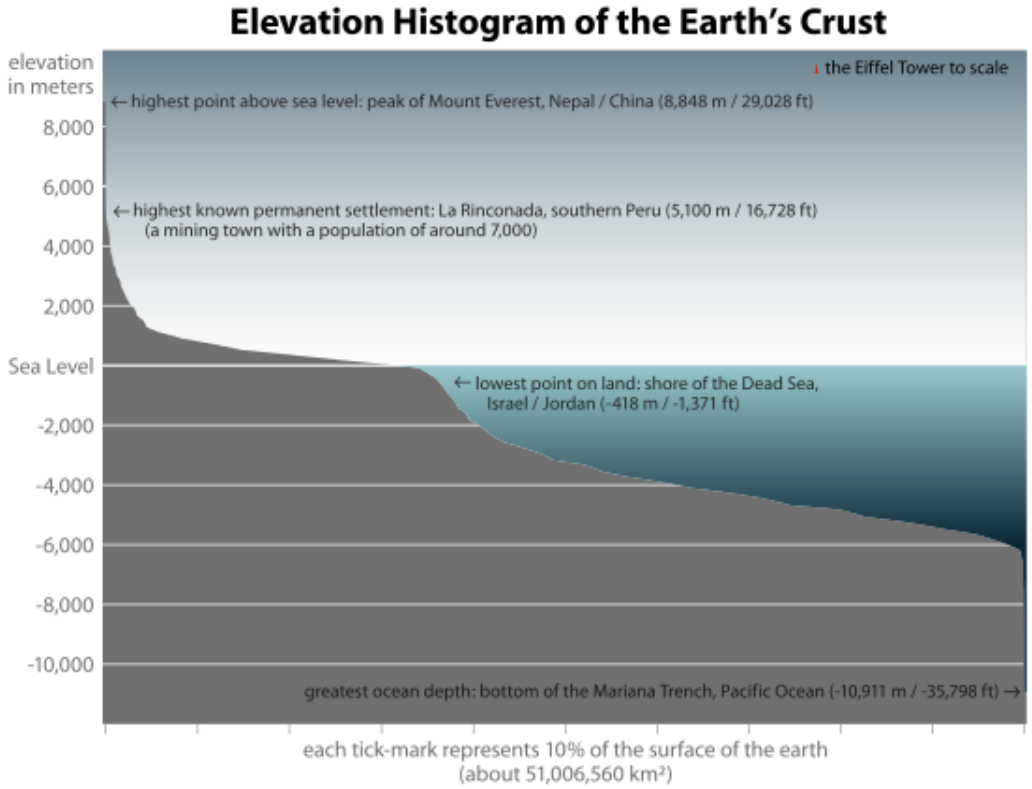
The Tree of Life (Source: Berkeley)

Geographic Coordinate System

Specify Terrestrial Location with 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.

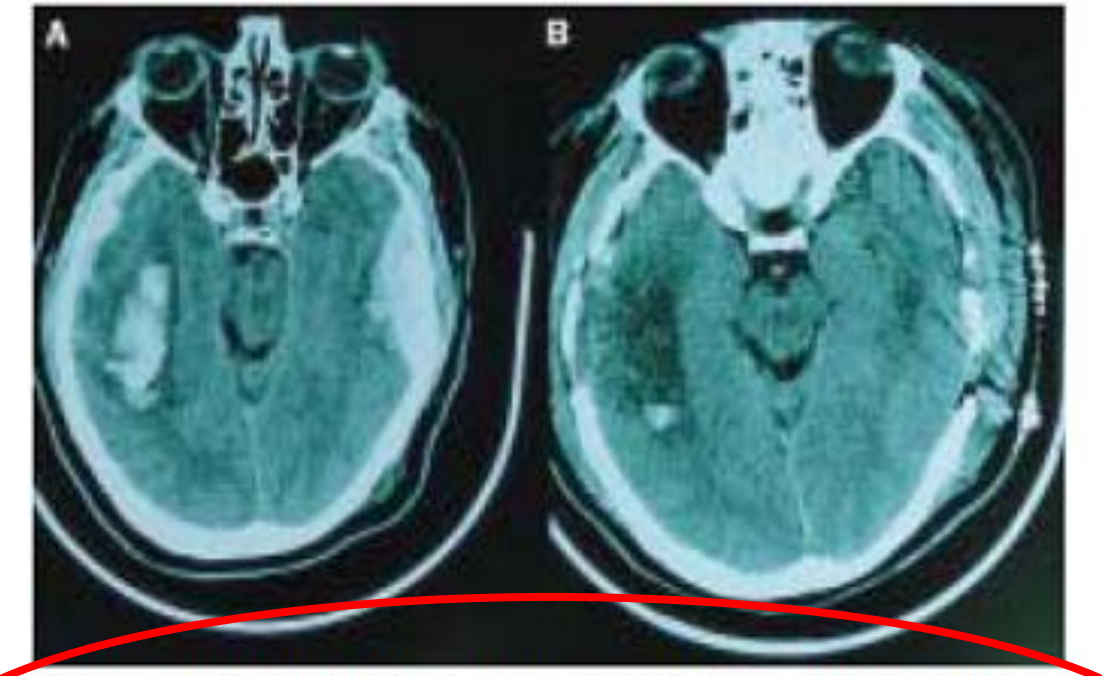


Figure 2: **Computed tomography of a comatose patient with a left temporal epidural haematoma, right parenchymal temporal lobe haematoma, and a right convexity subdural haematoma before and after craniotomy and evacuation of haematomas**

Outline

- Problems With Current Bug Descriptions
- Need for Structured Approach
- The Bugs Framework (BF)
- Examples of Applying BF
- Next Steps
- Benefits

The Bugs Framework (BF)

The Bugs Framework (BF)* will be
a precise descriptive language for bugs.

** BF is being created by factoring and restructuring of information contained in CWEs, SFPs STs, NSA CAS, SOAR, SEI-CERT and others and thus benefits from the community's experience with their use.*

What is the Bugs Framework (BF)?

- It is a set of classes of bugs.
- A BF bug class comprises:
  - Attributes that identify the software fault.
  - Causes that bring about the fault.
  - Consequences the fault could lead to.
  - Sites in code where the fault might occur.
- Causes and consequences are directed graphs.
- BF uses precise definitions and terminology.

First BF 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.
- Buffer Overflow (BOF).

→ Others under development.

BF: Buffer Overflow (BOF)

We started with Buffer Overflow.

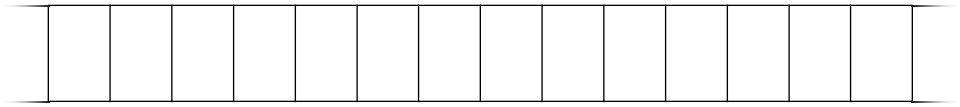
- Our Definition:
  The software accesses through an array a memory location that is outside the boundaries of that array.

- 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.”

BOF Attributes

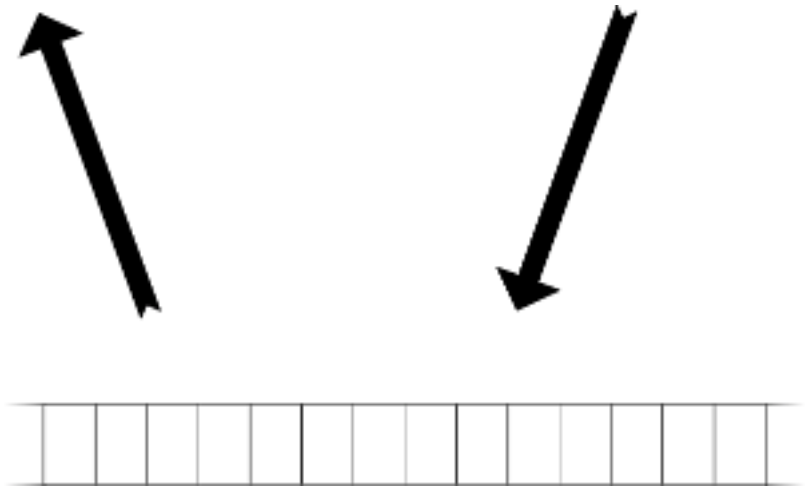
- Often referred to as a “buffer,” an array is a contiguously allocated set of objects, called elements.
  - Has a definite size – a definite number of elements are allocated to it.
  - Software should not use array name to access anything outside boundary of allocated elements.
  - Elements are all of same data type and accessed by integer offsets.
- If software can utilize array handle to access any memory other than allocated objects, it falls into this class.

An array could be pictured as follows:



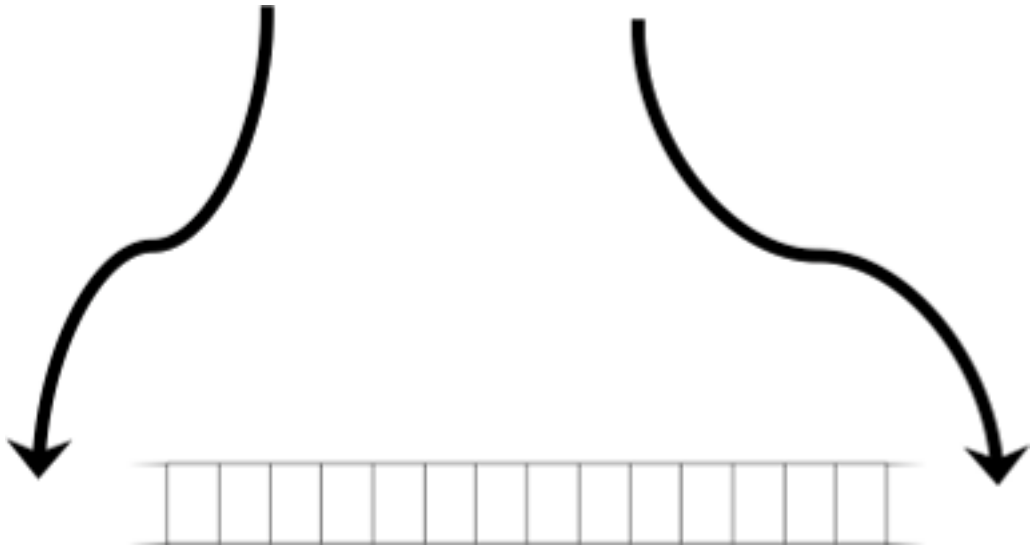
BOF Attributes – Access

- Access: Read, Write.



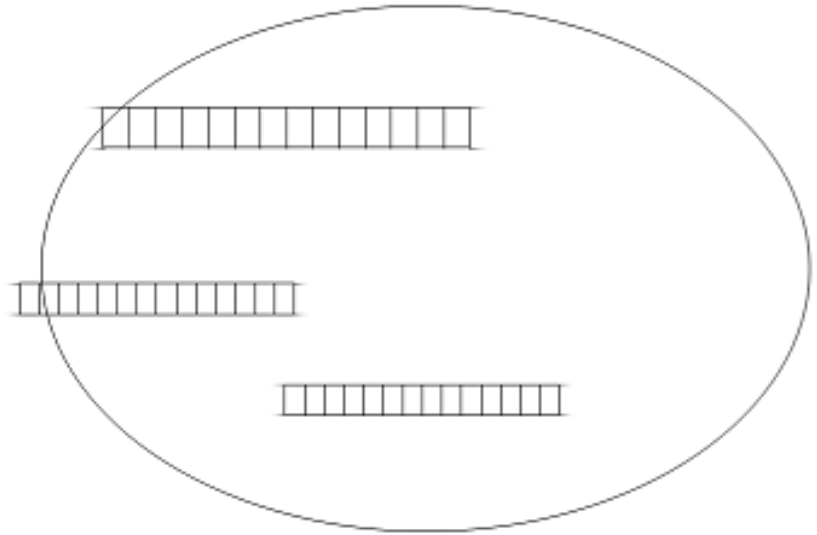
BOF Attributes – Boundary

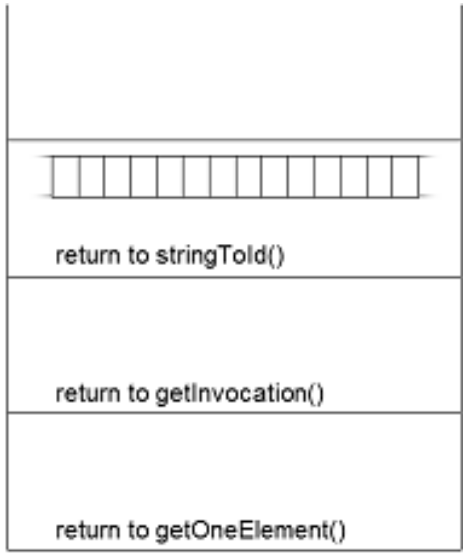
- Access: Read, Write.
- **Boundary** – which end of the array is violated:
  Below (before, under, or lower), Above (after, over, or upper).



BOF: Attributes – Location

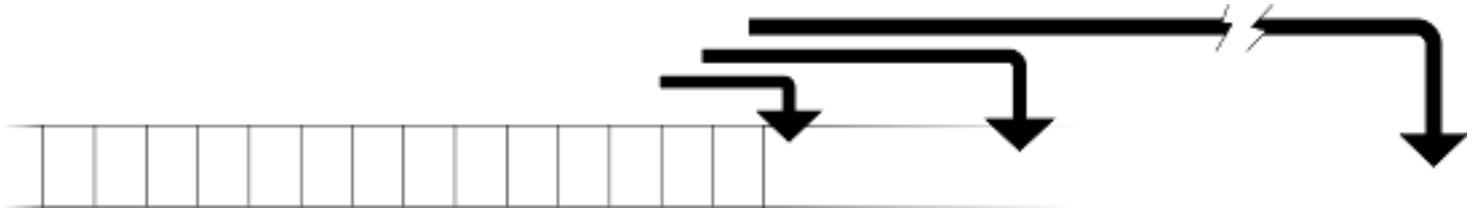
- Access: Read, Write.
- Boundary: Below, Above.
- **Location** – what part of memory the array is allocated in:
  Heap, Stack, BSS (uninitialized data), Data (initialized), Code (text).





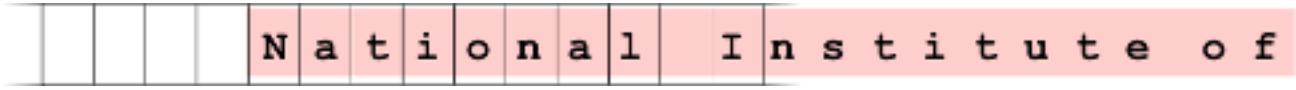
BOF Attributes – Magnitude

- Access: Read, Write.
- Boundary: Below, Above.
- Location: Heap, Stack, BSS, Data, Code.
- **Magnitude** – how far outside the boundary the violation extends:
  Small (just barely outside), Moderate (8 to dozens of bites), Far (e.g. 4000).



BOF Attributes – Data Size

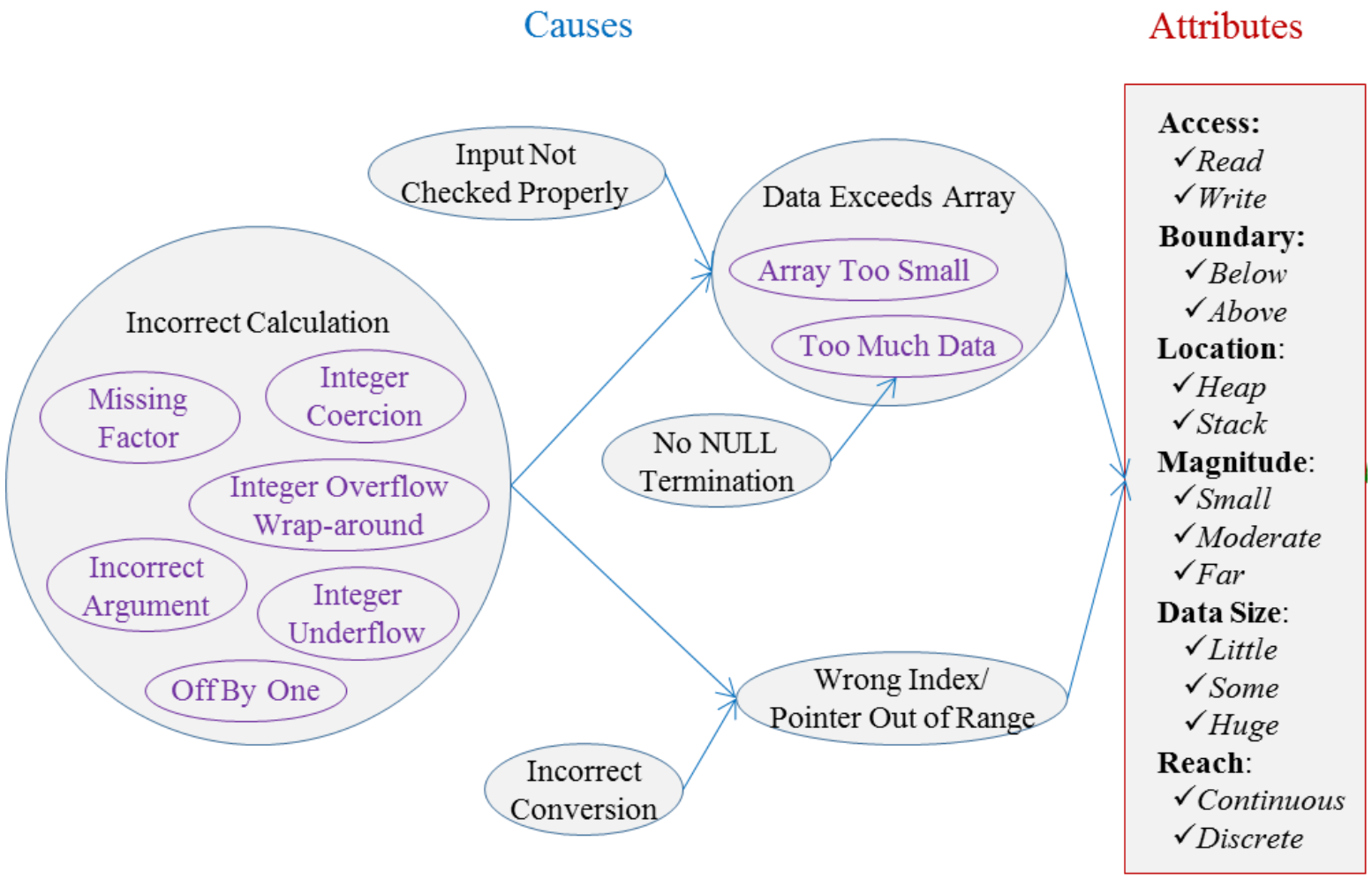
- Access: Read, Write.
- Boundary: Below, Above.
- Location: Heap, Stack, BSS, Data, Code.
- Magnitude: Small, Moderate, Far.
- **Data Size** – how much data is accessed beyond the boundary:
  Little, Some, Huge.



BOF Attributes – Reach

- Access: Read, Write.
- Boundary: Below, Above.
- Location: Heap, Stack, BSS, Data, Code.
- Magnitude: Small, Moderate, Far.
- Data Size: Little, Some, Huge.
- **Reach** – one-by-one or arbitrary:
  Continuous, Discrete.

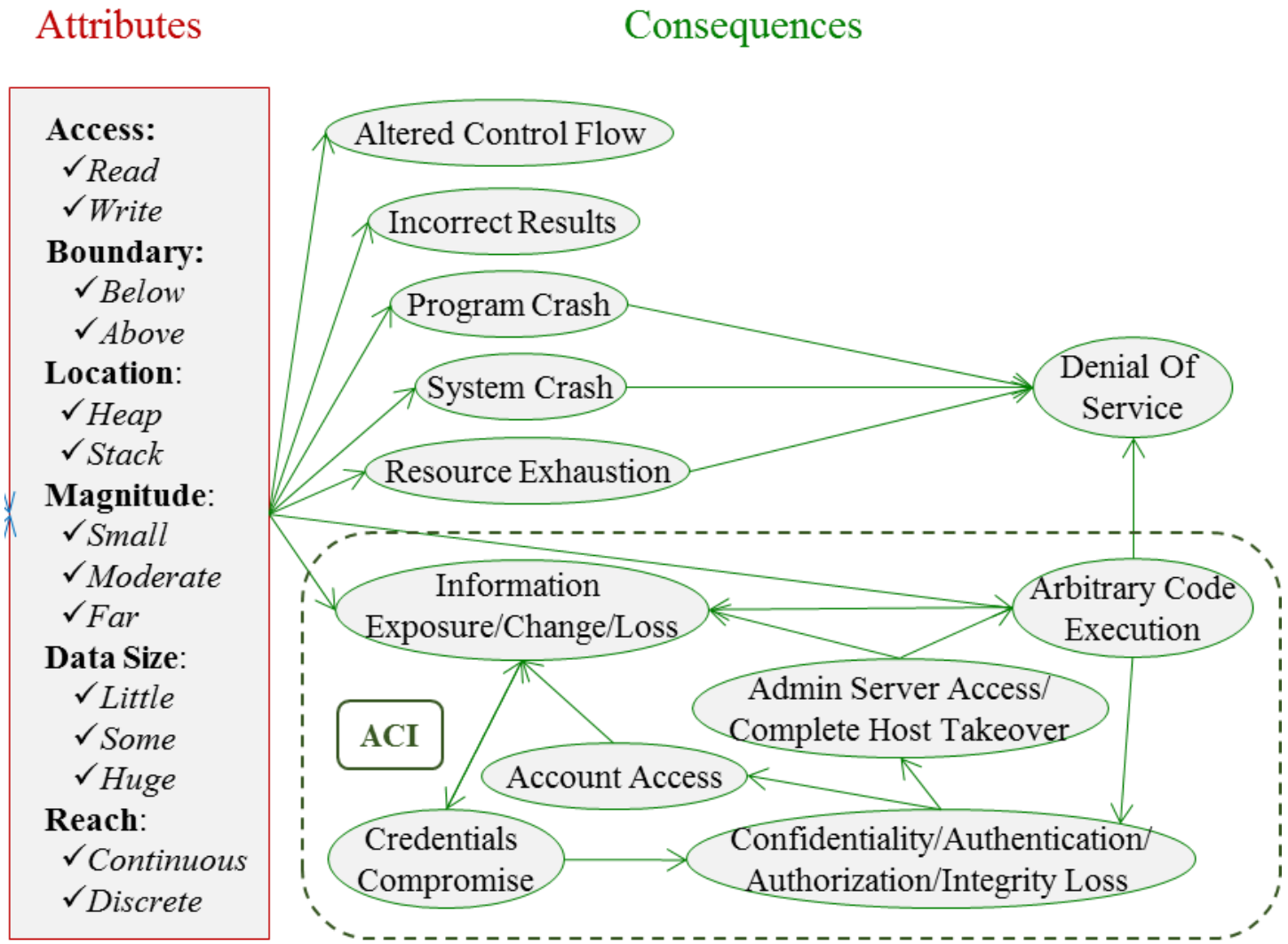
BOF: Causes



There are only two proximate causes of BOF with preceding causes that may lead to them:

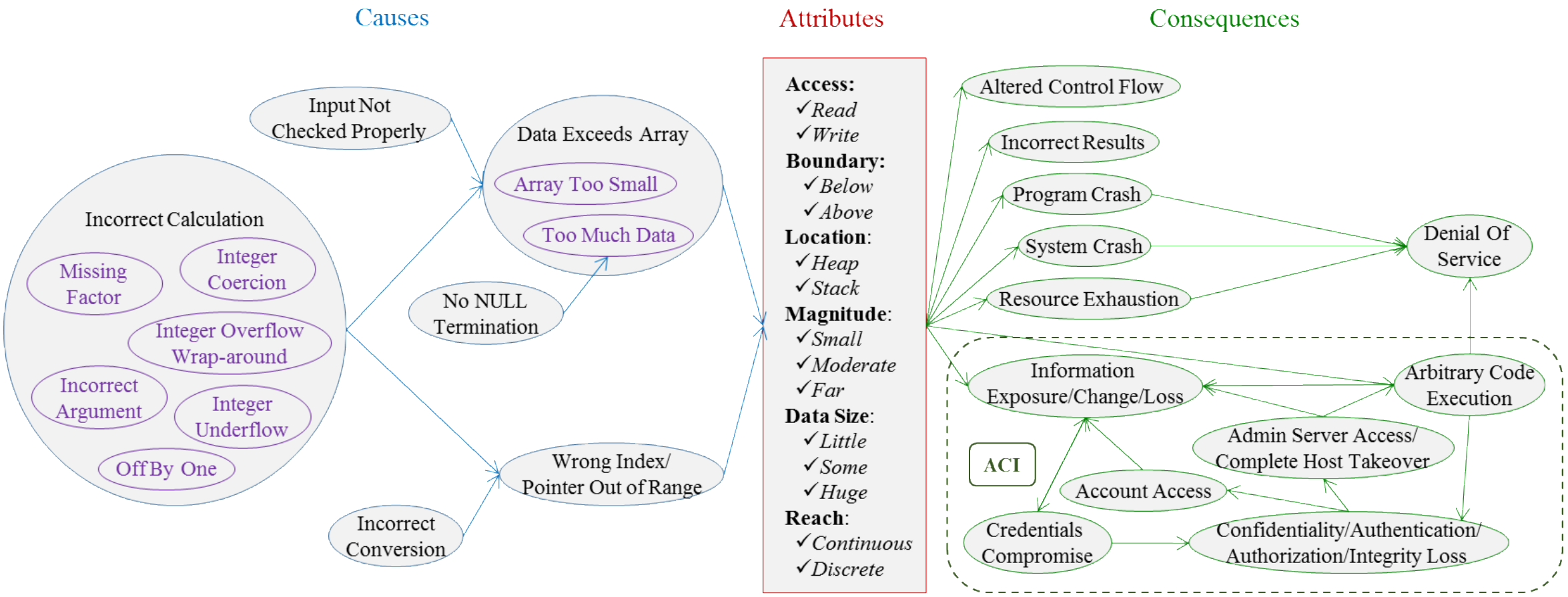
- Data Exceeds Array
- Wrong Index / Pointer Out of Range.

BOF: Consequences



Shows what could happen due to the fault.

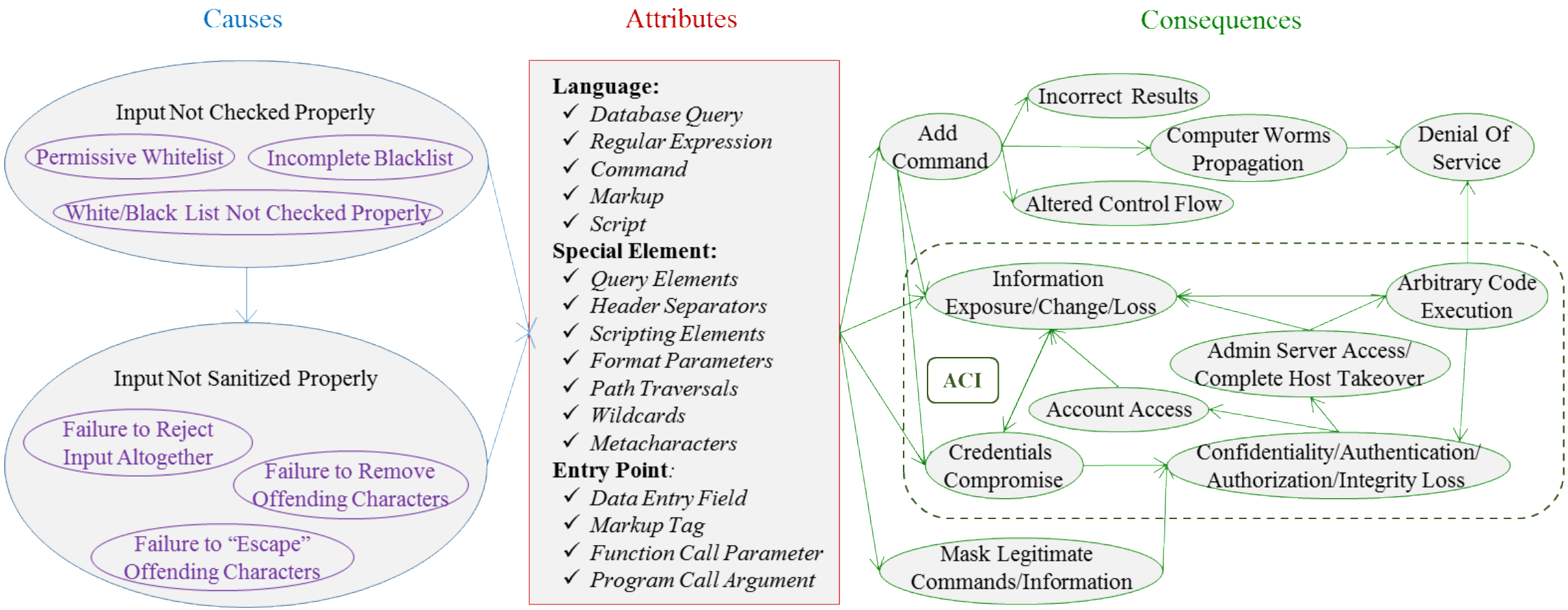
BOF: Causes, Attributes, and Consequences



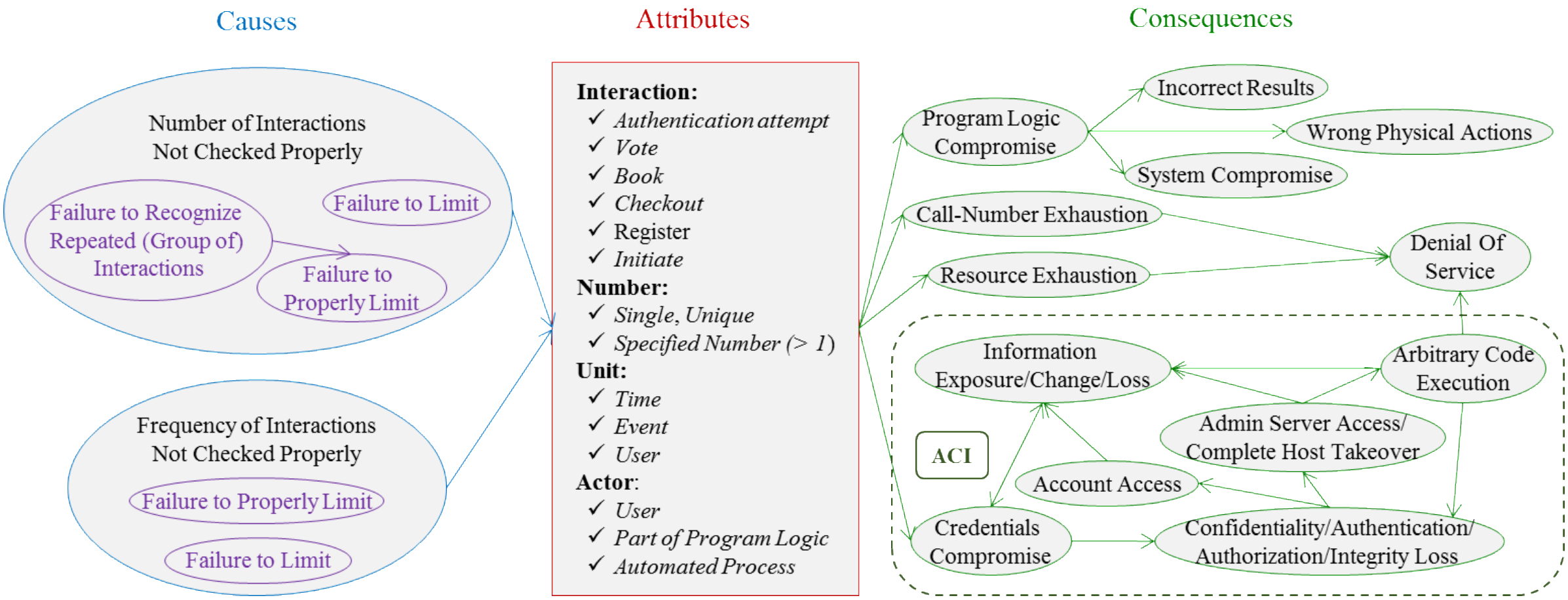
BOF: Sites

- Buffer Overflow may occur at:
  - ✓ use of [ ] operator with arrays in C
  - ✓ use of unary * operator with arrays in C
  - ✓ use of string library functions,
    such as strcpy() or strcat().

INJ: Causes, Attributes, and Consequences



CIF: Causes, Attributes, and Consequences



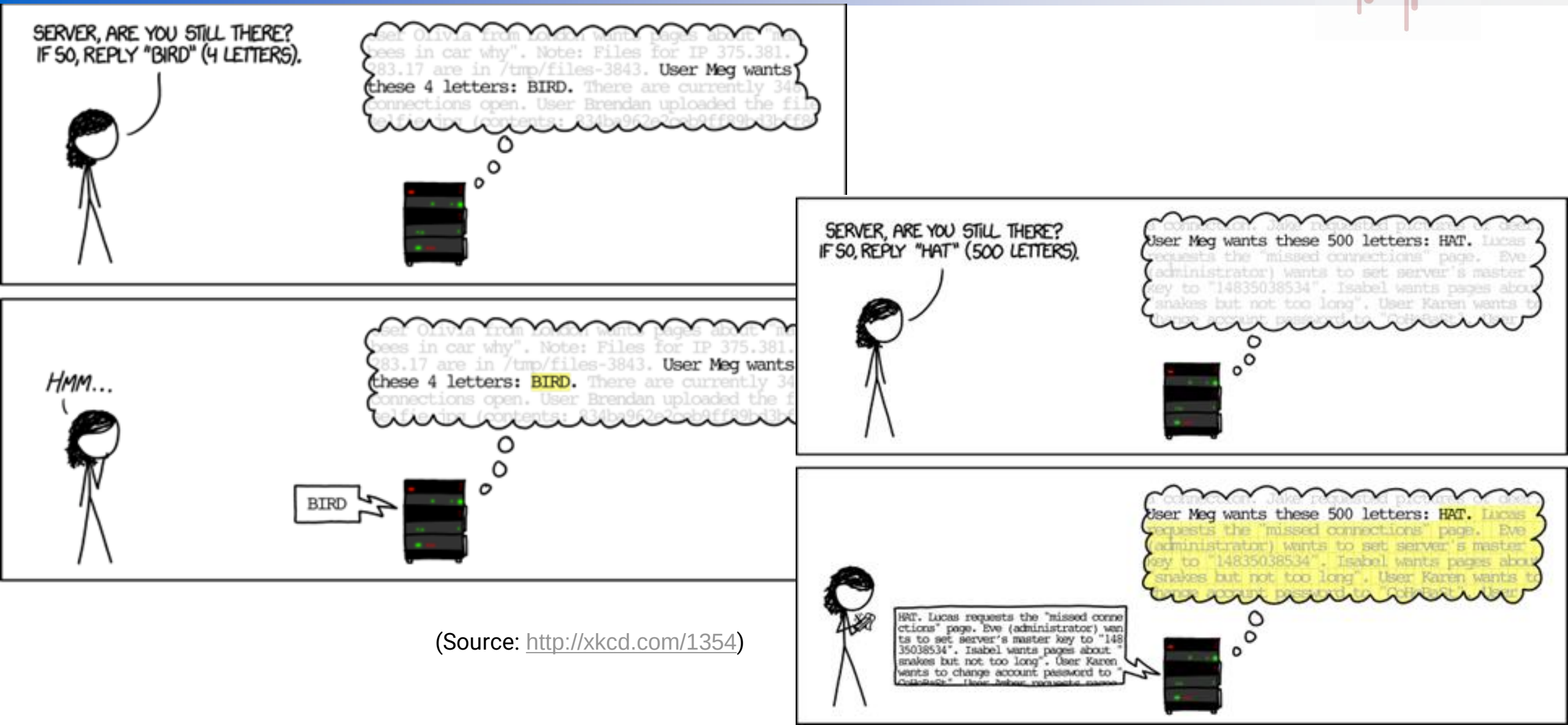
Outline

- Problems With Current Bug Descriptions
- Need for Structured Approach
- The Bugs Framework (BF)
- Examples of Applying BF
- Next Steps
- Benefits

Example 1: BF Explains Techniques

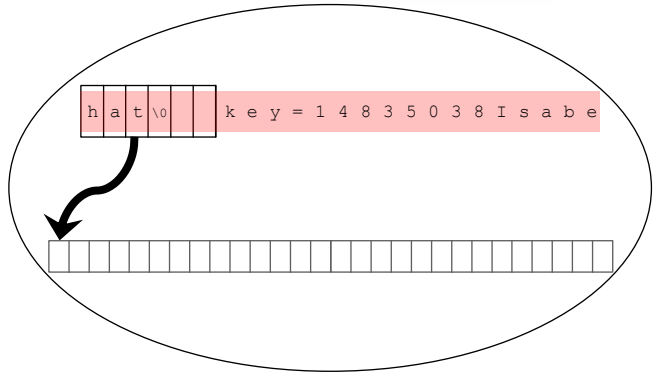
- Canaries
  - Extra memory above and below an array with unusual values, e.g., 0xDEADBEEF.
  - Useful with attributes:
    - Write *Access*
    - Small *Magnitude.*
- Address Space Layout Randomization (ASLR)
  - Allocate arrays randomly about memory.
  - Useful with attributes:
    - Heap *Location*
    - Stack *Location* – limited.

Example 2: Heartbleed



(Source: http://xkcd.com/1354)

Example 2: Heartbleed

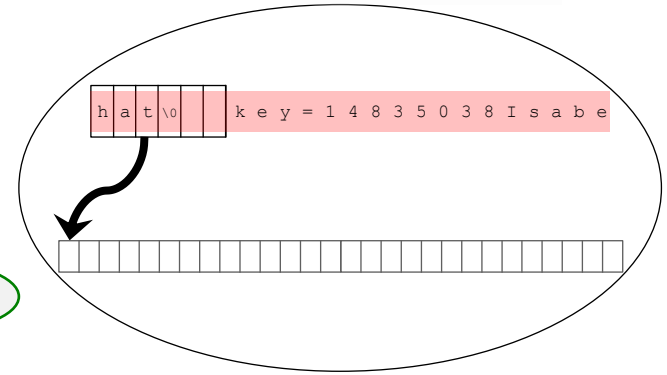


Heartbleed buffer overflow is:

- *Input Not Checked Properly* leads to
- *Data Exceeds Array* (specifically *Too Much Data*),
- where a *Huge* number of bytes
- are *Read* from the *Heap*
- in a *Continuous* reach
- *After* the array end,
- which may be exploited for *Exposure of Information* that had not been cleared *(CWE-226)*.

CVE-2014-0160 (Heartbleed): "*The (1) TLS and (2) DTLS implementations in OpenSSL 1.0.1 before 1.0.1g 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, related to d1_both.c and t1_lib.c, aka the Heartbleed bug.*"

Example 2: Heartbleed



Sensitive Info Uncleared Before Release

Input Not Checked Properly

Data Exceeds Array

Array Too Small

Too Much Data

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

Information Exposure

Incorrect Calculation

Missing Factor

Integer Coercion

Integer Overflow Wrap-around

Incorrect Argument

Integer Underflow

Off By One

No NULL Termination

Wrong Index / Pointer Out of Range

Incorrect Conversion

Information Change/Loss

Altered Control Flow

Incorrect Results

Resource Exhaustion

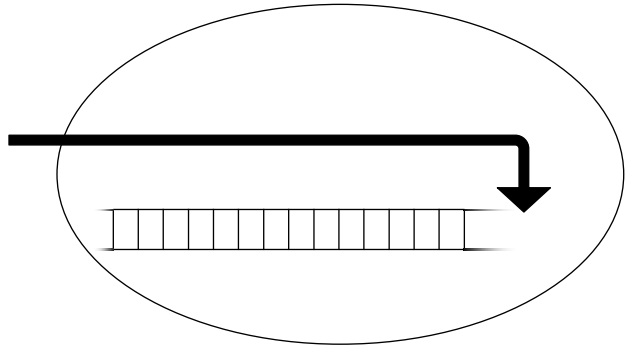
System Crash

Program Crash

Denial Of Service

Arbitrary Code Execution

Example 3: Ghost CVE-2015-0235

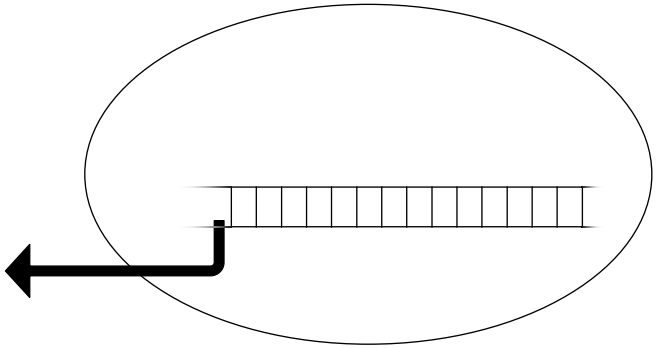


Ghost — gethostbyname buffer overflow is:

- *Incorrect Calculation*, (specifically *Missing Factor*) leads to
- *Data Exceeds Array* (specifically *Array Too Small*),
- where a *Moderate* number of bytes
- are *Written* to the *Heap*
- in a *Continuous* reach
- *After* the array end,
- which may be exploited for *Arbitrary Code Execution*, eventually leading to *Denial Of Service*.

CVE-2015-0235 (Ghost): *"Heap-based buffer overflow in the __nss_hostname_digits_dots function in glibc 2.2, and other 2.x versions before 2.18, allows context-dependent attackers to execute arbitrary code via vectors related to the (1) gethostbyname or (2) gethostbyname2 function, aka GHOST."*

Example 4: Chrome CVE-2010-1773



Chrome WebCore — render buffer overflow is:

- *Incorrect Calculation*, (specifically *Off By One*) leads to
- a *Wrong Index*,
- where a *Small* number of bytes
- are *Read* from the *Heap*
- in a *Discrete* reach
- *Before* the array start, which may be exploited for *Information Exposure*, *Arbitrary Code Execution* or *Program Crash*,
  leading to *Denial Of Service*.

CVE-2010-1773 (Chrome WebCore): *"Off-by-one error in the toAlphabetic function in rendering/RenderListMarker.cpp in WebCore in WebKit before r59950, as used in Google Chrome before 5.0.375.70, 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, aka rdar problem 8009118."*

Example 5: Refactoring CWEs

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

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				

Outline

- Problems With Current Bug Descriptions
- Need for Structured Approach
- The Bugs Framework (BF)
- Examples of Applying BF
- Next Steps
- Benefits

Next Steps

- Continue with the thorough research on:
  - ✓ Programming languages
  - ✓ Programming practices, and
  - ✓ Available repositories (enumerations/patterns/templates/standards/projects) of software weaknesses, vulnerabilities and attacks: CWEs, SFPs, STs, NSA CAS, SOAR, SEI-CERT and others.
- Engage stakeholders from the federal government, US industry and academia, and hold meetings to understand the current state-of-the-art in this field.
- Reason, factor, enrich and structure the gathered information into an orthogonal complete hierarchical classification of software bugs.
- Build out all BF classes.

.

Next Steps: Bugs Areas

- Software Weaknesses Areas:
  - Access:
    - ✓ Authentication
    - ✓ Authorization
    - ✓ Permissions and Privileges.
  - Functionality:
    - ✓ Expressions: Calculations, Comparisons, Functions
    - ✓ Control Flow: Branching, Looping, Concurrency, Race Conditions
    - ✓ Exceptions.
  - Data (used, stored, transmitted):
    - ✓ Memory (+ Initialization)
    - ✓ Files & Directories
    - ✓ Communications.

Upcoming: BF Access Classes

- BF Classes related to Access:
  - ✓ ATN (Authentication)
    - Make sure entity is who they claim to be.
  - ✓ AUT (Authorization – often conflated with Access Control)
    - Make sure entity is allowed to perform the operation.
  - ✓ CRY (Cryptography)
    - ENC (Encryption) is only one part of CRY.
    - Is IEX a part of CRY?
  - ✓ RND (Randomization)
    - Random Number Generation, etc.
  - ✓ CIF (Control of Interaction Frequency) – has ATN.

Upcoming: BF Functionality Classes

- BF Classes related to Functionality:
  - ✓ FOP (Faulty Operation) – Calculations, Comparisons, Functions, Cast
    - Attributes:
      Operator (add, Shift, Cast, Call (that is, parameter passing during call))
    - Operand fault (mismatched data types, overflow - value too big, etc.) – This is typically the relationship between “operands” of the operation, not just the characteristic of one operand.
    - Result Fault (why result is wrong) (value becomes small, large, undefined, etc.)
    - Data Type (int, floating point, pointer, etc.) – This is a sub- or meta-attribute. For instance, mismatch data type could be int -> pointer

    IOF (Integer Overflow)
    DBZ (Divide by Zero), …
  - ✓ FLO (Control Flow) – Branching, Looping (Switch without default, Infinite loop,...)
  - ✓ INJ (Injection)
  - ✓ EXC (Exception handling)
    – Throw, Try, Catch
  - ✓ CON (Concurrency)
    – Deadlock, Starvation (unfair scheduling), Races, Locks, Synchronization, etc.

Upcoming: BF Data Classes

- BF Classes related to Data:
  - ✓ MEM (Memory+Initialization – data in use): Use after free, Memory leak.
    - Memory is usually just a giant array, maybe with allocation and freeing.
    - Memory is non-persistent.
      - BOF (Buffer Overflow) – ?BFR (Read) – is a BOF; BFW (Write) – is a BOF?
  - ✓ STO (Storage/File System – data at rest)
    - Storage is typically intricately structured, that is, with a file system. Access is largely by means of the file system with all its names, permissions, links, etc.
    - Storage is generally persistent - one thinks of files as lasting far longer than processes.
  - ✓ NET (Network – data in transit)
    - Network is significantly different from memory and storage.

✓ IEX (Information Exposure) – has an ENC.

Next Steps: Hierarchy, Graphs & Mappings

- Hierarchy of abstract & concrete classes of bugs with:
  - ✓ Precise Definitions
  - ✓ Attributes that identify or distinguish the software fault
- Directed graph(s) of Causes that bring about faults.
- Directed graph(s) of Consequences faults could lead to.
- Causes-Attributes-Consequences Mappings.
- Sites in code where the fault might occur.

Note: BF uses precise terminology.

Next Steps: Refinement

- Investigate consequences - it almost seems like everything can cause anything.
  - ✓ Maybe consequences should be finer grained.
  - ✓ Maybe they need context.
- More precise (formal?) definitions of causes and consequences.
- Include code excerpts
- Include context? e.g. off-by-one for BOF only causes *small* magnitude overflows.
- Come up with a succinct nomenclature for attributes.

- Write BF descriptions for many CWEs
  - *and get MITRE to add a new field for them.*
- More examples. In particular, can we describe tool warning classes well?
- Website (guide books or encyclopedia):
  record why we chose this instead of that, the example of consequence A leading to B, etc.
- Incorporate CVE info? file and function name, line numbers, etc.

Next Steps: Dissemination and Use

- Incorporate in current bugs and vulnerabilities enumerations
  – such as CWE and CVE.
- Incorporate in existing and new software assurance tools
  – enhance them to report in terms of BF descriptions.

- Summarize results in a NIST IR.
- Disseminate through publications and a dedicated website (https://samate.nist.gov/BF).
- Eventually, push BF forward as a standard.

Outline

- Problems With Current Bug Descriptions
- Need for Structured Approach
- The Bugs Framework (BF)
- Examples of Applying BF
- Next Steps
- Benefits

Benefits

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

- Describe problems in software and discuss the classes of bugs that tools 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.

Benefits

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

- Precisely and unambiguously express software bugs or vulnerabilities.
- Estimate risk and determine best mitigation strategies based on known consequences of different kinds of faults.
- 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 general kind of fault that has other attributes.

Questions

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