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/

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

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.

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 The Networking and Information Technology Research and Development Program

Search NITRD.gov

HOME | ABOUT NITRD | COORDINATION AREAS | NEWS & EVENTS | PUBLICATIONS & RESOURCES | CONTACT US

Software Productivity, Sustainability, and Quality (SPSQ IWG)

The purpose of the SPSQ IWG is to coordinate the R&D efforts across agencies that transform the frontiers of software science and engineering and to identify R&D areas in need of development that span the science and the technology of software creation and sustainment.

NITRD -> NITRD GROUPS -> *SPSQ IWG*

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

NITRD The Networking and Information Technology Research and Development Program

Search NITRD.gov

HOME | ABOUT NITRD | COORDINATION AREAS | NEWS & EVENTS | PUBLICATIONS & RESOURCES | CONTACT US

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

Cyber Security and Information Assurance (CSIA) Interagency Working Group coordinates the activities of the CSIA Program Component Area.

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:

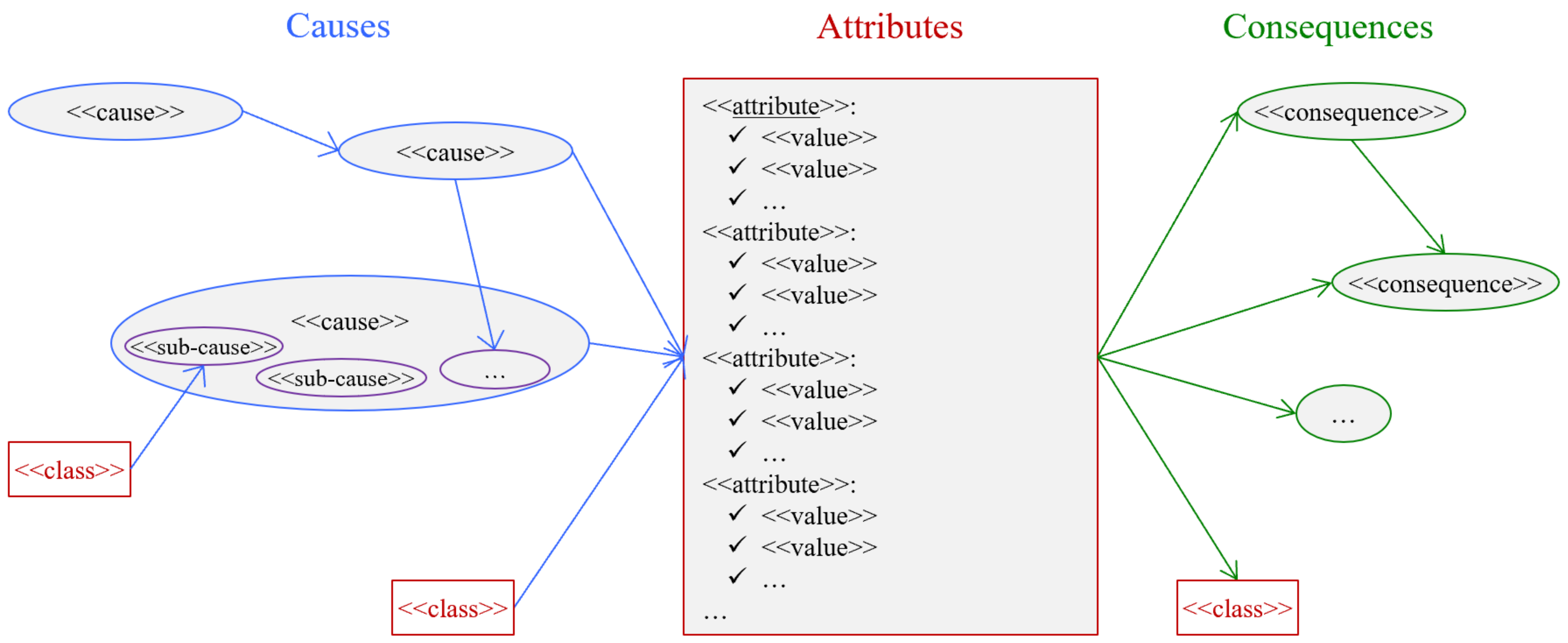
✓ Level (high or low)	– identifies the fault as language-related or semantic.
✓ Attributes	– identify the software fault.
✓ Causes	– bring about the fault.
✓ Consequences	– to which the fault could lead.
✓ Sites	– locations in code where the fault might occur.

○ Sites are identifiable mainly for low level classes

→ BF uses precise definitions and terminology.

- BF is *descriptive*, not *prescriptive*.
  - ✓ 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:

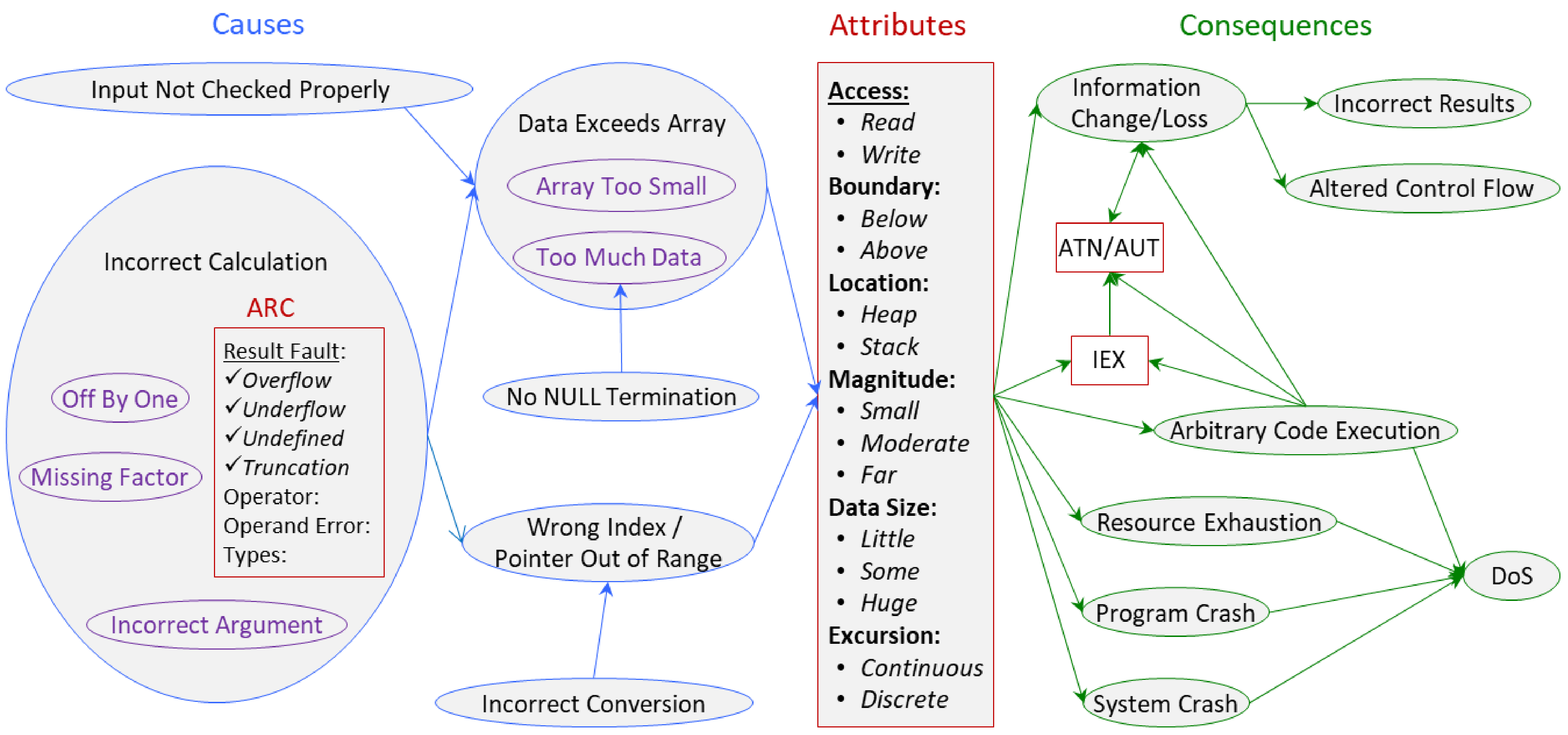
  The software accesses through an array a memory location that is outside the boundaries of that array.

  → 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



Result Fault:
✓*Overflow*
✓*Underflow*
✓*Undefined*
✓*Truncation*
Operator:
Operand Error:
Types:

Access:
- *Read*
- *Write*

Boundary:
- *Below*
- *Above*

Location:
- *Heap*
- *Stack*

Magnitude:
- *Small*
- *Moderate*
- *Far*

Data Size:
- *Little*
- *Some*
- *Huge*

Excursion:
- *Continuous*
- *Discrete*

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)`

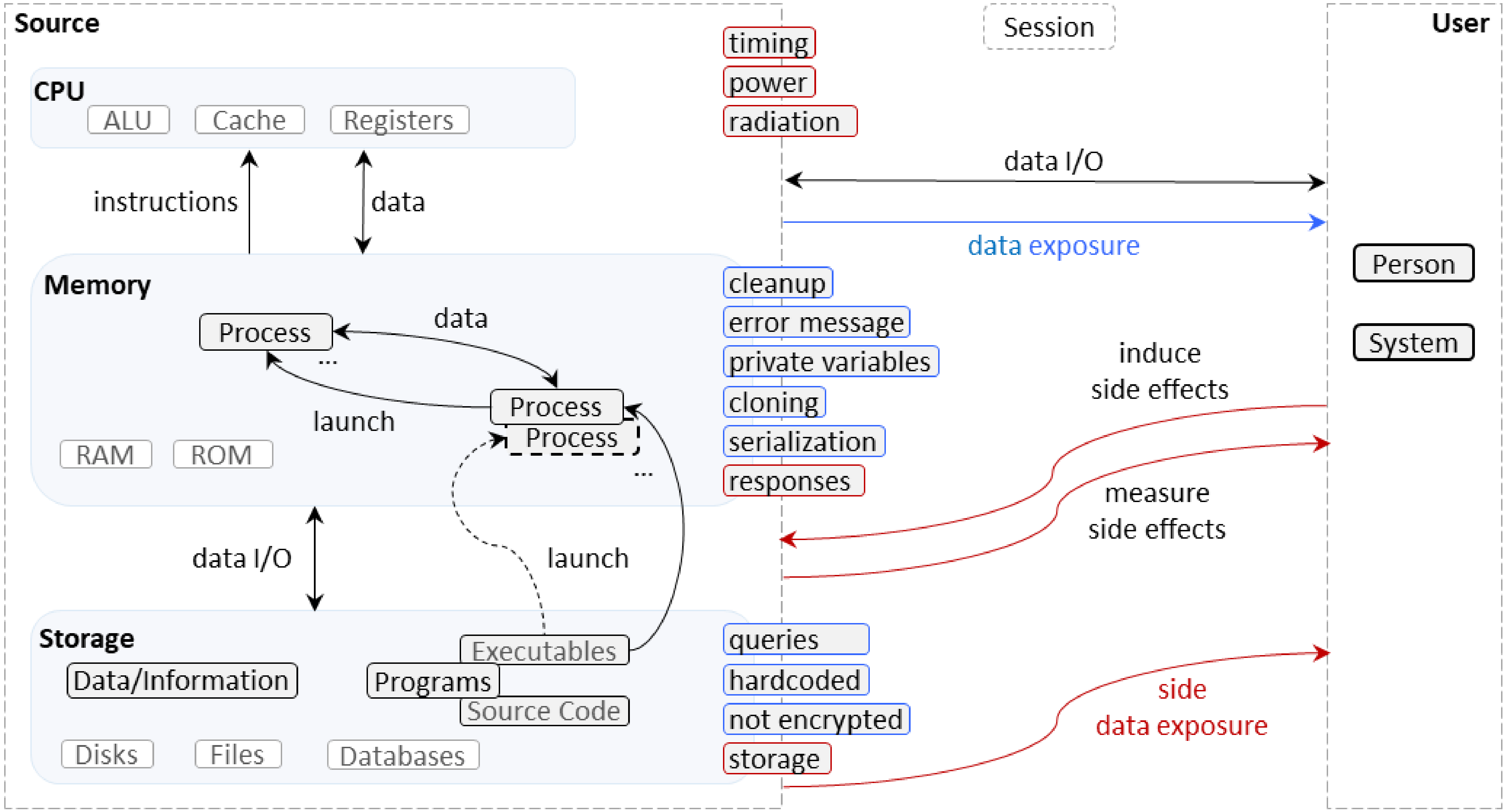
- Using all the attributes: `144 (=2x2x2x3x3x2)`
  Using all the attributes, the 3 direct causes, and 6 consequences, without constraints:
  Total `2592 (= 3x(2x2x2x3x3x2)x6)`

- 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.

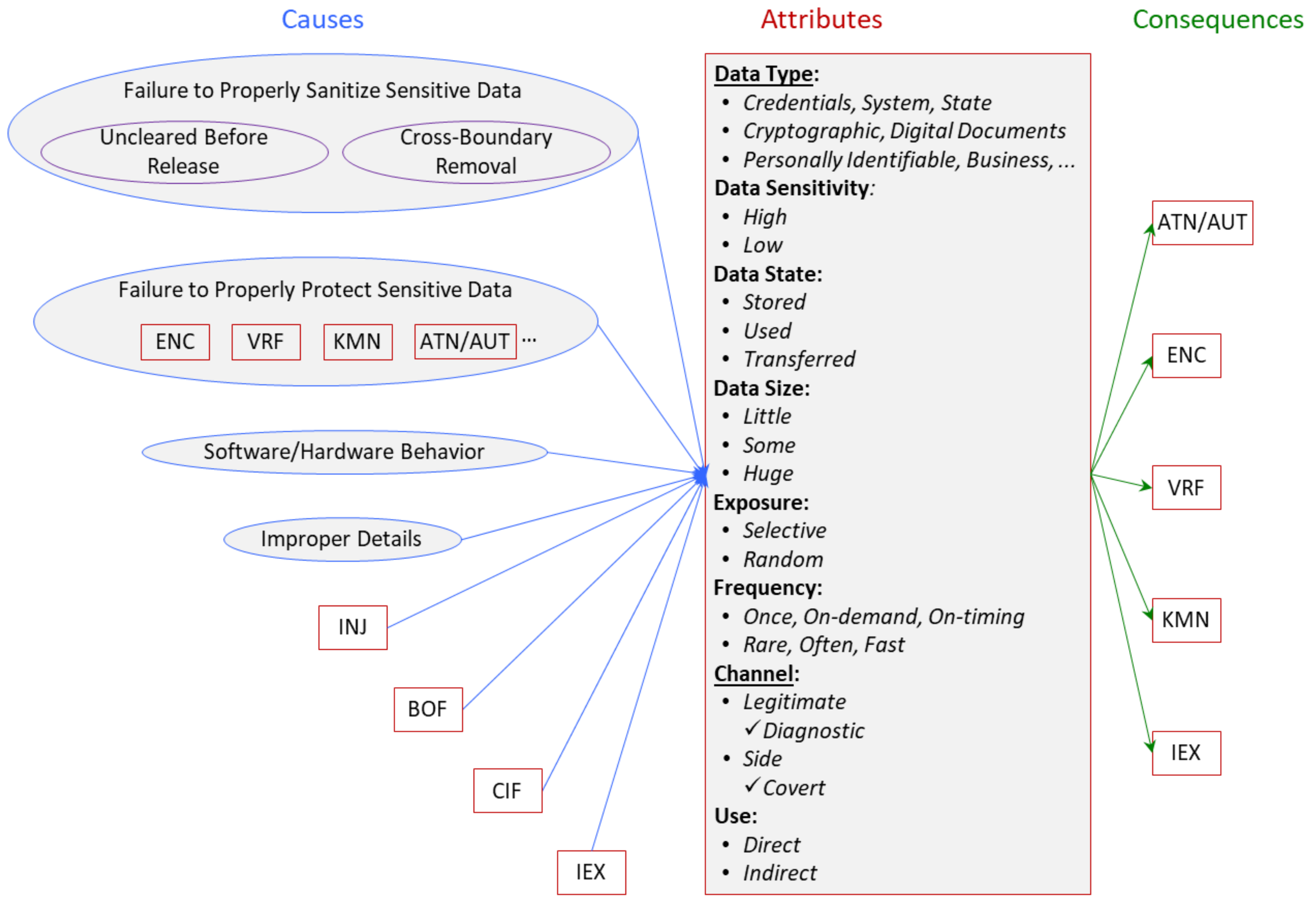
Related CWEs and SFPs:

- 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



Data Type:
- *Credentials, System, State*
- *Cryptographic, Digital Documents*
- *Personally Identifiable, Business, …*

Data Sensitivity*:*
- *High*
- *Low*

Data State:
- *Stored*
- *Used*
- *Transferred*

Data Size:
- *Little*
- *Some*
- *Huge*

Exposure:
- *Selective*
- *Random*

Frequency:
- *Once, On-demand, On-timing*
- *Rare, Often, Fast*

Channel:
- *Legitimate*
  - ✓*Diagnostic*
- *Side*
  - ✓*Covert*

Use:
- *Direct*
- *Indirect*

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>>".

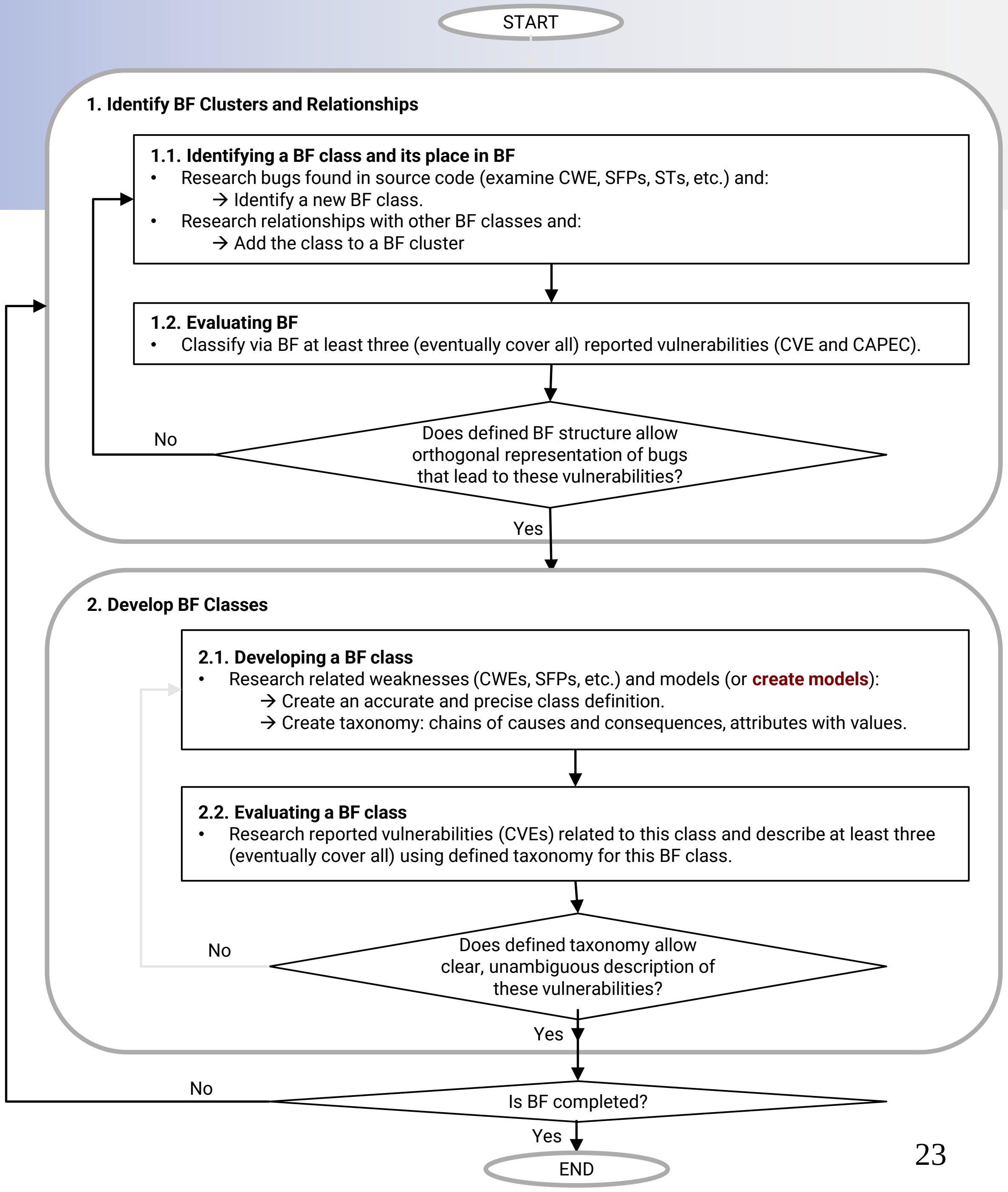
BF Class Taxonomy:

- **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"



START
1. Identify BF Clusters and Relationships
1.1. Identifying a BF class and its place in BF
• Research bugs found in source code (examine CWE, SFPs, STs, etc.) and:
→ Identify a new BF class.
• Research relationships with other BF classes and:
→ Add the class to a BF cluster
1.2. Evaluating BF
• Classify via BF at least three (eventually cover all) reported vulnerabilities (CVE and CAPEC).
No
Does defined BF structure allow orthogonal representation of bugs that lead to these vulnerabilities?
Yes
2. Develop BF Classes
2.1. Developing a BF class
• Research related weaknesses (CWEs, SFPs, etc.) and models (or create models):
→ Create an accurate and precise class definition.
→ Create taxonomy: chains of causes and consequences, attributes with values.
2.2. Evaluating a BF class
• Research reported vulnerabilities (CVEs) related to this class and describe at least three (eventually cover all) using defined taxonomy for this BF class.
No
Does defined taxonomy allow clear, unambiguous description of these vulnerabilities?
Yes
No
Is BF completed?
Yes
END

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.

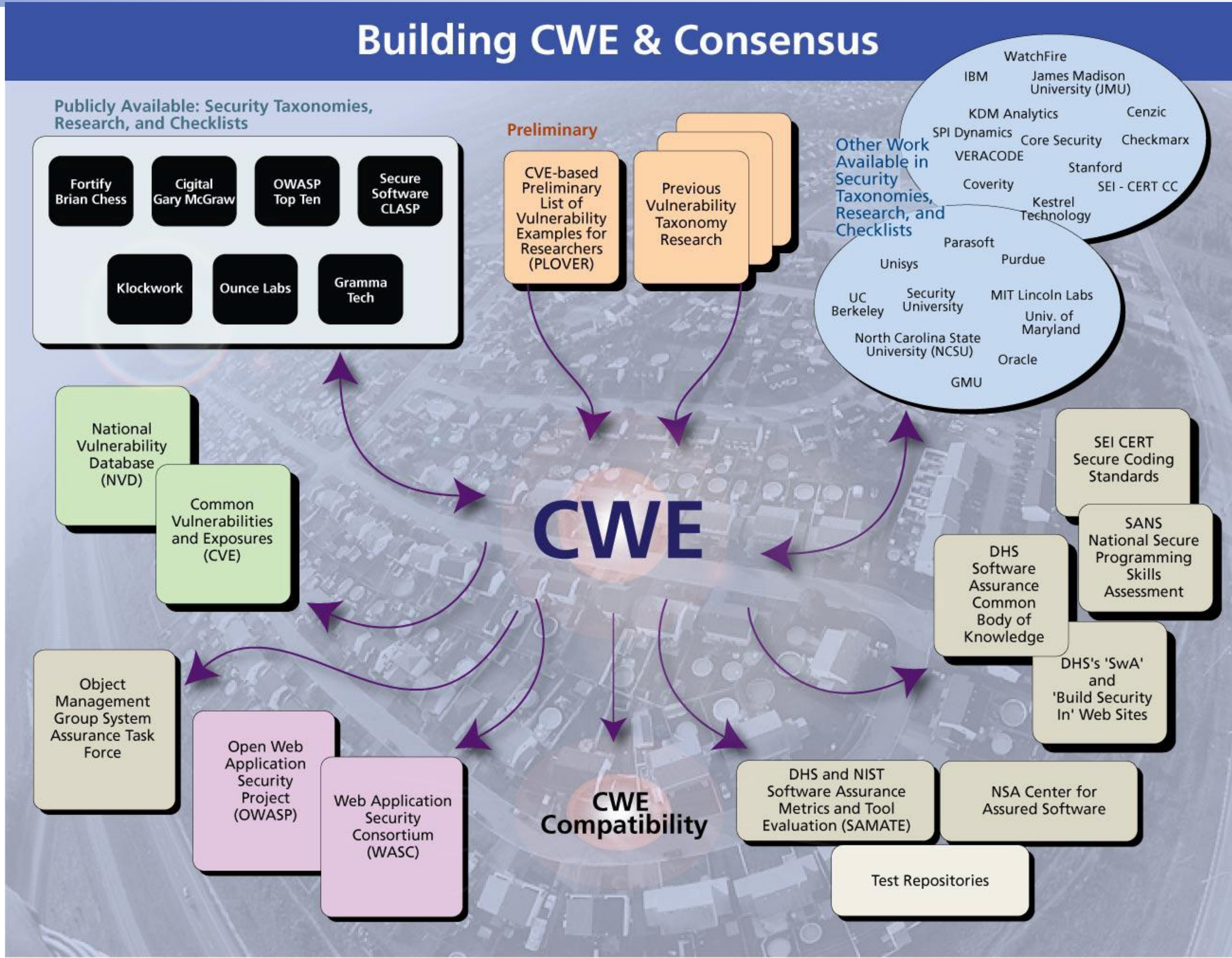

Building CWE & Consensus
Publicly Available: Security Taxonomies, Research, and Checklists
Fortify Brian Chess
Cigital Gary McGraw
OWASP Top Ten
Secure Software CLASP
Klockwork
Ounce Labs
Gramma Tech
Preliminary
CVE-based Preliminary List of Vulnerability Examples for Researchers (PLOVER)
Previous Vulnerability Taxonomy Research
Other Work Available in Security Taxonomies, Research, and Checklists
WatchFire
IBM
James Madison University (JMU)
KDM Analytics
Cenzic
SPI Dynamics
Core Security
Checkmarx
VERACODE
Stanford
Coverity
SEI - CERT CC
Kestrel Technology
Parasoft
Unisys
Purdue
UC Berkeley
Security University
MIT Lincoln Labs
Univ. of Maryland
North Carolina State University (NCSU)
Oracle
GMU
National Vulnerability Database (NVD)
Common Vulnerabilities and Exposures (CVE)
CWE
SEI CERT Secure Coding Standards
SANS National Secure Programming Skills Assessment
DHS Software Assurance Common Body of Knowledge
DHS's 'SwA' and 'Build Security In' Web Sites
Object Management Group System Assurance Task Force
Open Web Application Security Project (OWASP)
Web Application Security Consortium (WASC)
CWE Compatibility
DHS and NIST Software Assurance Metrics and Tool Evaluation (SAMATE)
NSA Center for Assured Software
Test Repositories

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.

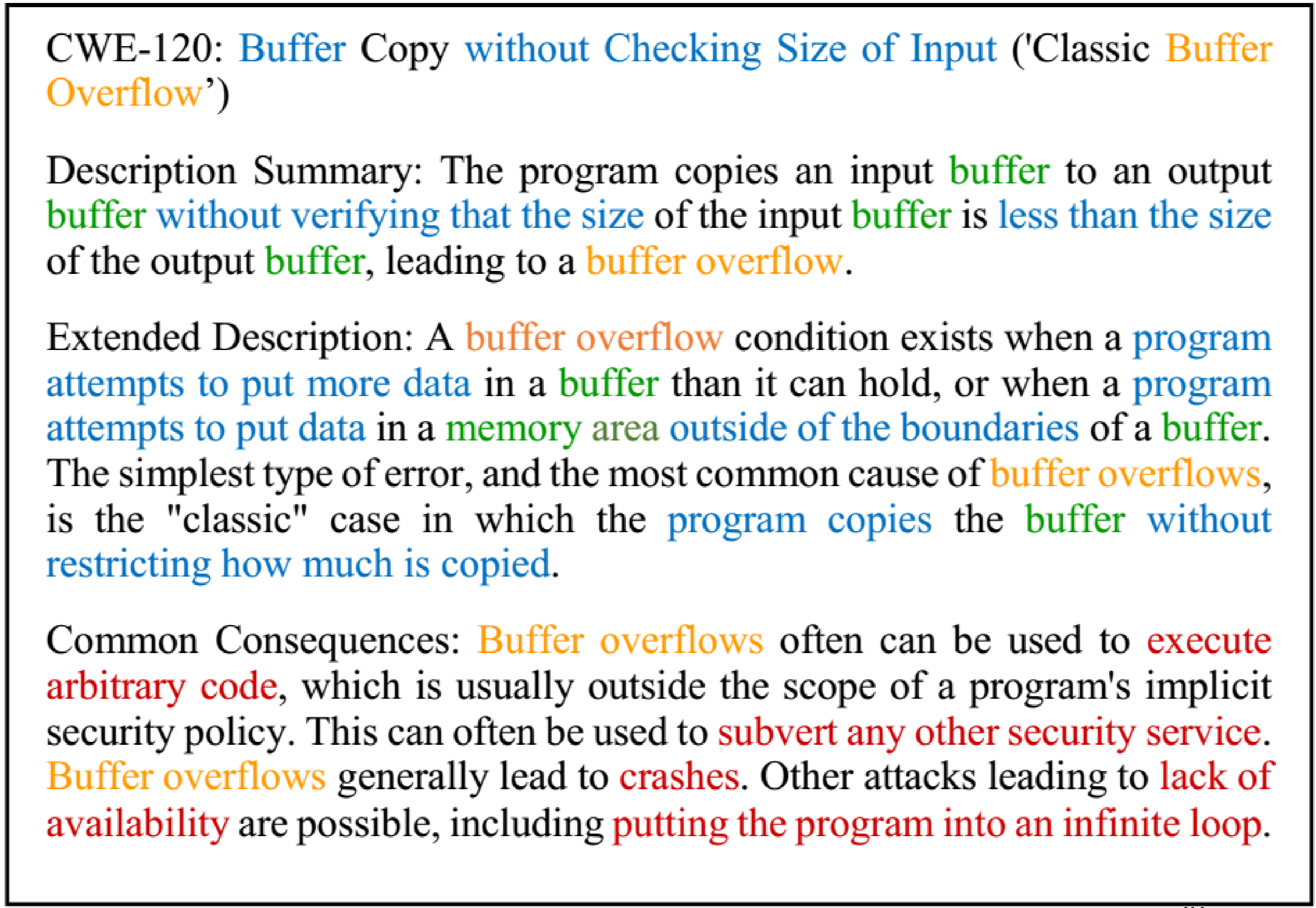
Semantic Templates (ST)

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.

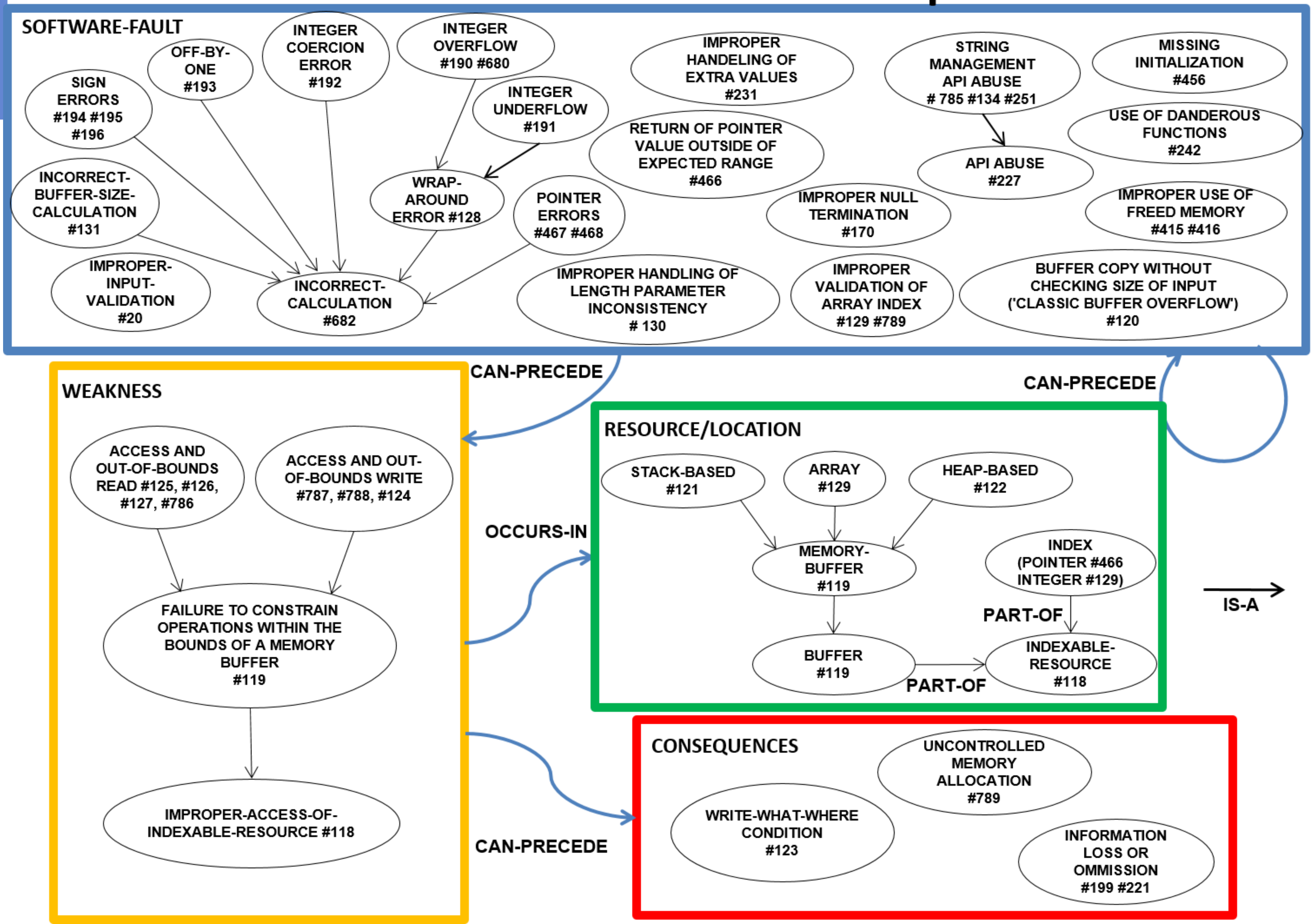
Common Consequences: Buffer overflows often can be used to execute arbitrary code, which is usually outside the scope of a program's implicit security policy. This can often be used to subvert any other security service. Buffer overflows generally lead to crashes. Other attacks leading to lack of availability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

Buffer Overflow Semantic Template



SOFTWARE-FAULT
SIGN ERRORS #194 #195 #196
OFF-BY-ONE #193
INTEGER COERCION ERROR #192
INTEGER OVERFLOW #190 #680
INTEGER UNDERFLOW #191
IMPROPER HANDELING OF EXTRA VALUES #231
STRING MANAGEMENT API ABUSE # 785 #134 #251
MISSING INITIALIZATION #456
USE OF DANDEROUS FUNCTIONS #242
API ABUSE #227
RETURN OF POINTER VALUE OUTSIDE OF EXPECTED RANGE #466
INCORRECT-BUFFER-SIZE-CALCULATION #131
WRAP-AROUND ERROR #128
POINTER ERRORS #467 #468
IMPROPER NULL TERMINATION #170
IMPROPER USE OF FREED MEMORY #415 #416
IMPROPER-INPUT-VALIDATION #20
INCORRECT-CALCULATION #682
IMPROPER HANDLING OF LENGTH PARAMETER INCONSISTENCY # 130
IMPROPER VALIDATION OF ARRAY INDEX #129 #789
BUFFER COPY WITHOUT CHECKING SIZE OF INPUT ('CLASSIC BUFFER OVERFLOW') #120
CAN-PRECEDE
CAN-PRECEDE
WEAKNESS
ACCESS AND OUT-OF-BOUNDS READ #125, #126, #127, #786
ACCESS AND OUT-OF-BOUNDS WRITE #787, #788, #124
FAILURE TO CONSTRAIN OPERATIONS WITHIN THE BOUNDS OF A MEMORY BUFFER #119
IMPROPER-ACCESS-OF-INDEXABLE-RESOURCE #118
OCCURS-IN
RESOURCE/LOCATION
STACK-BASED #121
ARRAY #129
HEAP-BASED #122
MEMORY-BUFFER #119
INDEX (POINTER #466 INTEGER #129)
PART-OF
BUFFER #119
PART-OF
INDEXABLE-RESOURCE #118
IS-A
CAN-PRECEDE
CONSEQUENCES
UNCONTROLLED MEMORY ALLOCATION #789
WRITE-WHAT-WHERE CONDITION #123
INFORMATION LOSS OR OMMISSION #199 #221

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

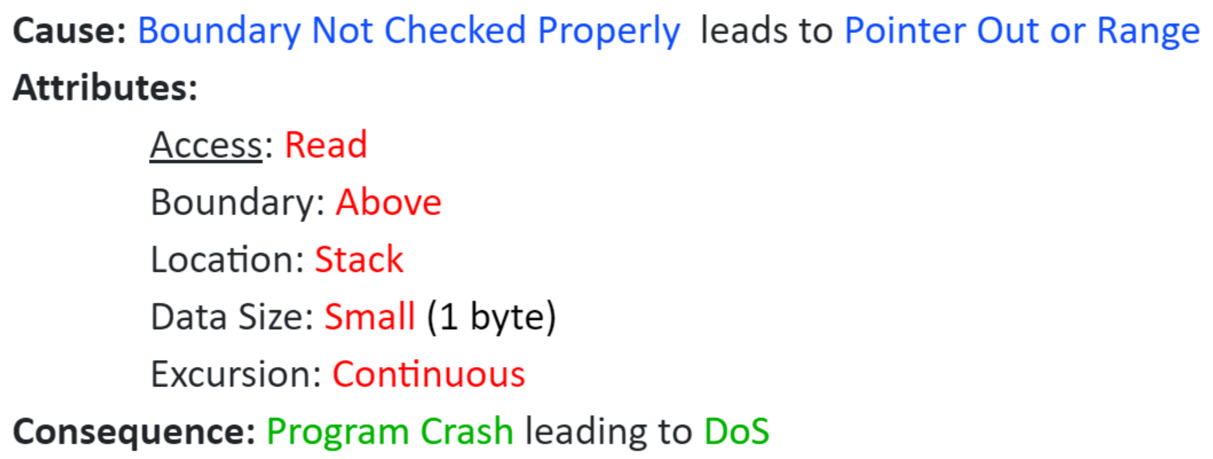
The empty cells in the table show the overlaps and gaps in the CWEs coverage of buffer overflow options with the following attributes considered:

- ✓ 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*')**

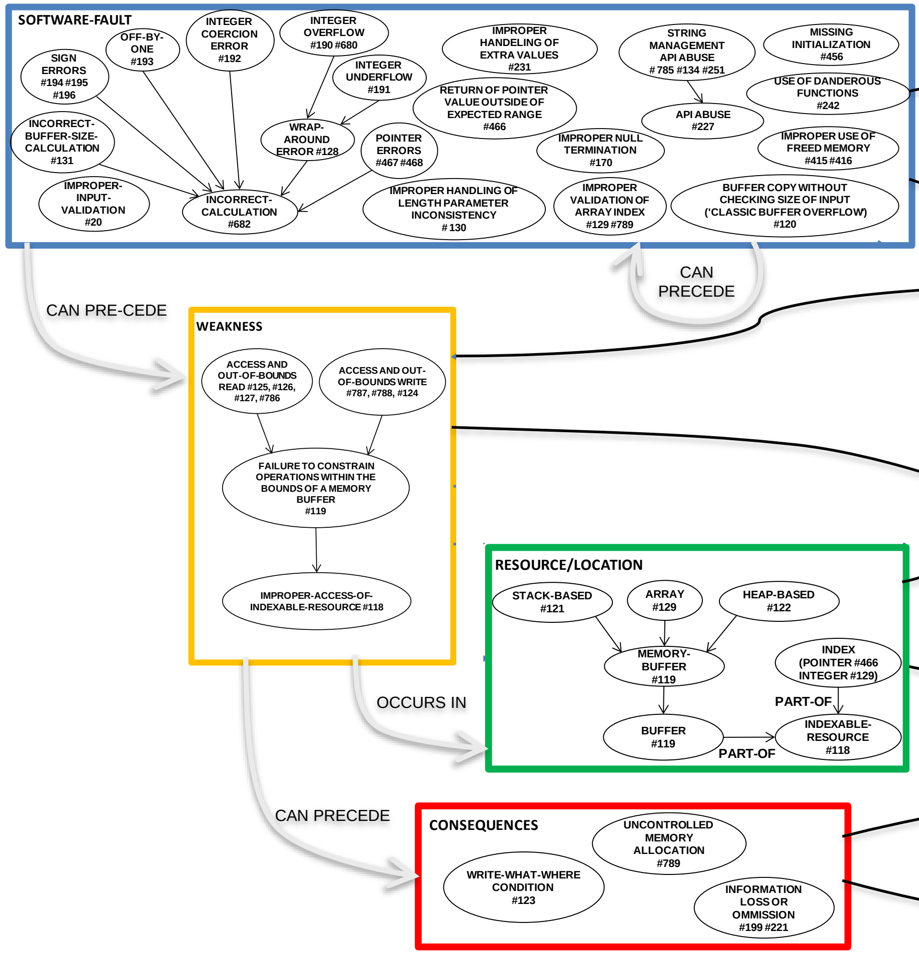
Summary: The program copies an input ***buffer*** to an output ***buffer*** ***without verifying that the size of the input buffer is less than the size of the output buffer***, leading to a ***buffer overflow***.

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



SOFTWARE-FAULT
SIGN ERRORS #194 #195 #196
OFF-BY-ONE #193
INTEGER COERCION ERROR #192
INTEGER OVERFLOW #190 #680
INTEGER UNDERFLOW #191
IMPROPER HANDELING OF EXTRA VALUES #231
STRING MANAGEMENT API ABUSE # 785 #134 #251
MISSING INITIALIZATION #456
USE OF DANDEROUS FUNCTIONS #242
RETURN OF POINTER VALUE OUTSIDE OF EXPECTED RANGE #466
API ABUSE #227
INCORRECT-BUFFER-SIZE-CALCULATION #131
WRAP-AROUND ERROR #128
POINTER ERRORS #467 #468
IMPROPER NULL TERMINATION #170
IMPROPER USE OF FREED MEMORY #415 #416
IMPROPER-INPUT-VALIDATION #20
INCORRECT-CALCULATION #682
IMPROPER HANDLING OF LENGTH PARAMETER INCONSISTENCY # 130
IMPROPER VALIDATION OF ARRAY INDEX #129 #789
BUFFER COPY WITHOUT CHECKING SIZE OF INPUT ('CLASSIC BUFFER OVERFLOW) #120
CAN PRECEDE
CAN PRE-CEDE
WEAKNESS
ACCESS AND OUT-OF-BOUNDS READ #125, #126, #127, #786
ACCESS AND OUT-OF-BOUNDS WRITE #787, #788, #124
FAILURE TO CONSTRAIN OPERATIONS WITHIN THE BOUNDS OF A MEMORY BUFFER #119
IMPROPER-ACCESS-OF-INDEXABLE-RESOURCE #118
OCCURS IN
RESOURCE/LOCATION
STACK-BASED #121
ARRAY #129
HEAP-BASED #122
MEMORY-BUFFER #119
INDEX (POINTER #466 INTEGER #129)
PART-OF
BUFFER #119
PART-OF
INDEXABLE-RESOURCE #118
CAN PRECEDE
CONSEQUENCES
UNCONTROLLED MEMORY ALLOCATION #789
WRITE-WHAT-WHERE CONDITION #123
INFORMATION LOSS OR OMMISSION #199 #221

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.

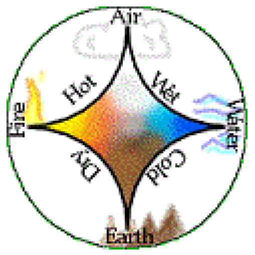
- As a result the newly delivered and the legacy systems will continue having security holes despite all the patching to correct errant behavior.

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

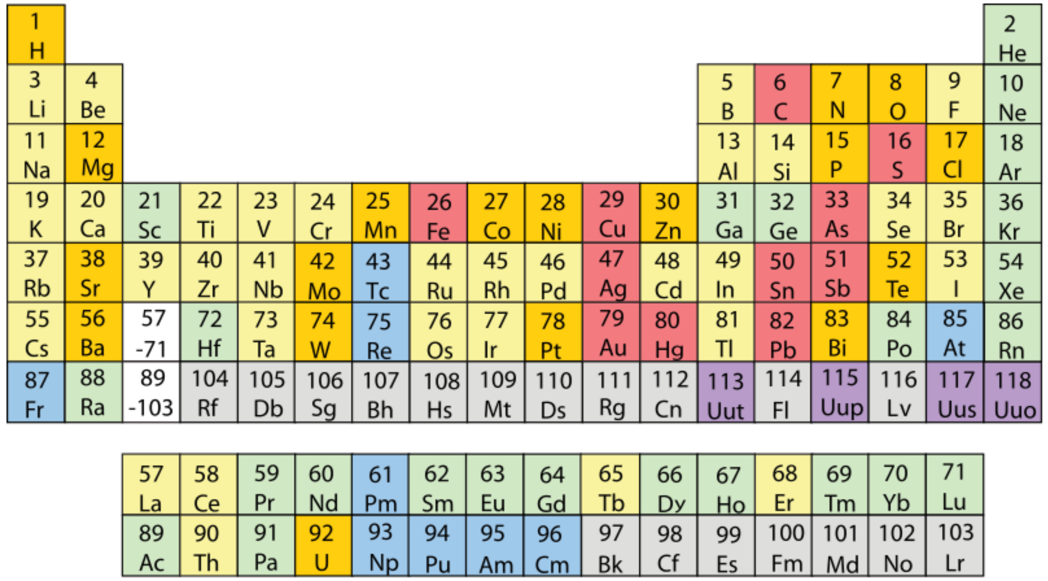
But, for analogies on what we are embarking on, let’s look at
some well-know organizational structures in science …

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)

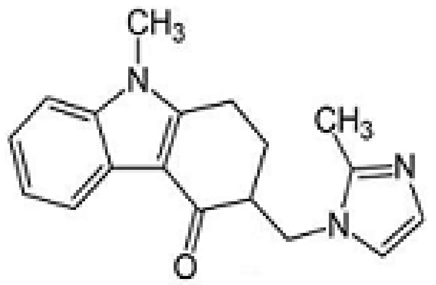


1 H																	2 He
3 Li	4 Be											5 B	6 C	7 N	8 O	9 F	10 Ne
11 Na	12 Mg											13 Al	14 Si	15 P	16 S	17 Cl	18 Ar
19 K	20 Ca	21 Sc	22 Ti	23 V	24 Cr	25 Mn	26 Fe	27 Co	28 Ni	29 Cu	30 Zn	31 Ga	32 Ge	33 As	34 Se	35 Br	36 Kr
37 Rb	38 Sr	39 Y	40 Zr	41 Nb	42 Mo	43 Tc	44 Ru	45 Rh	46 Pd	47 Ag	48 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

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.

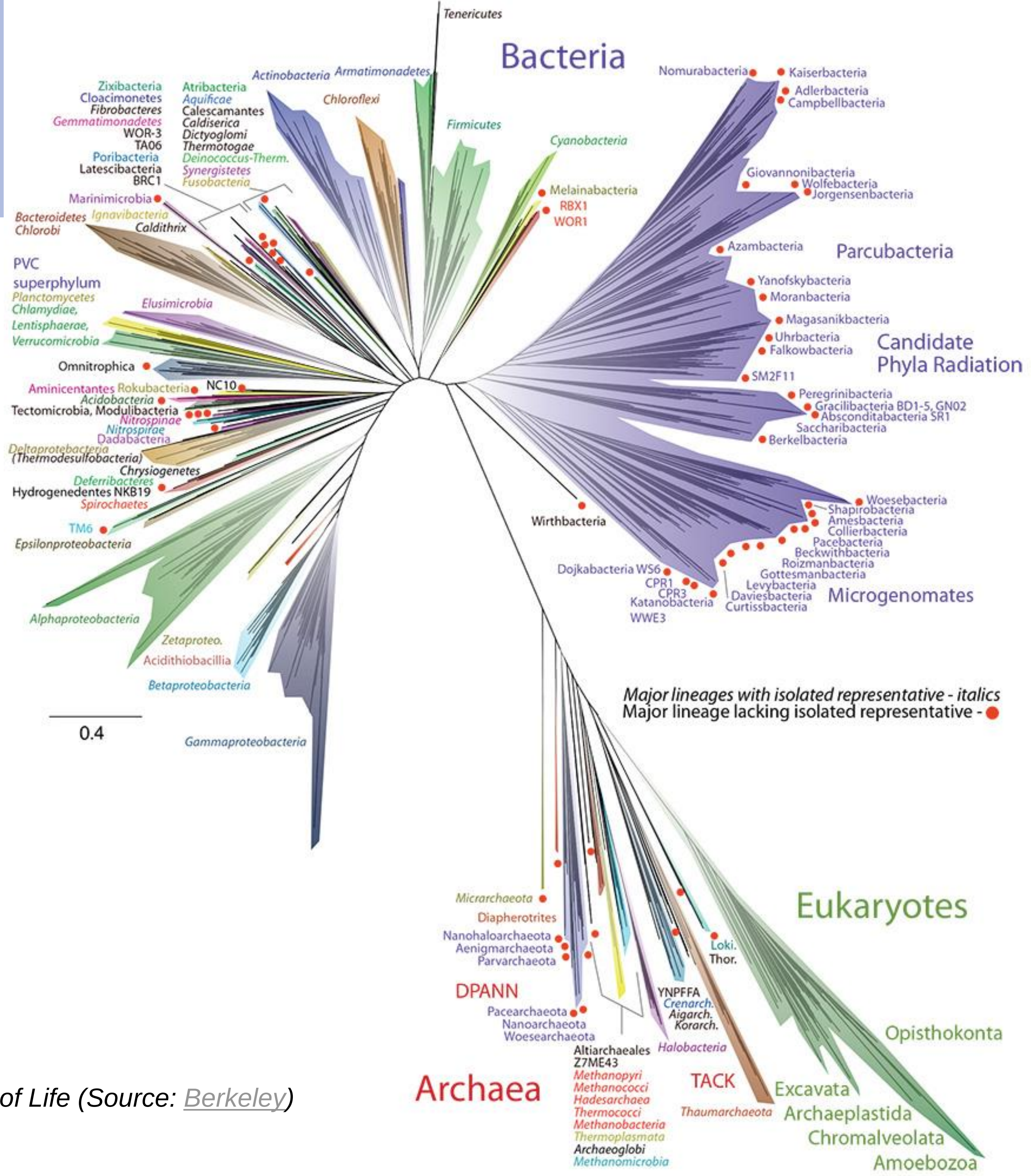
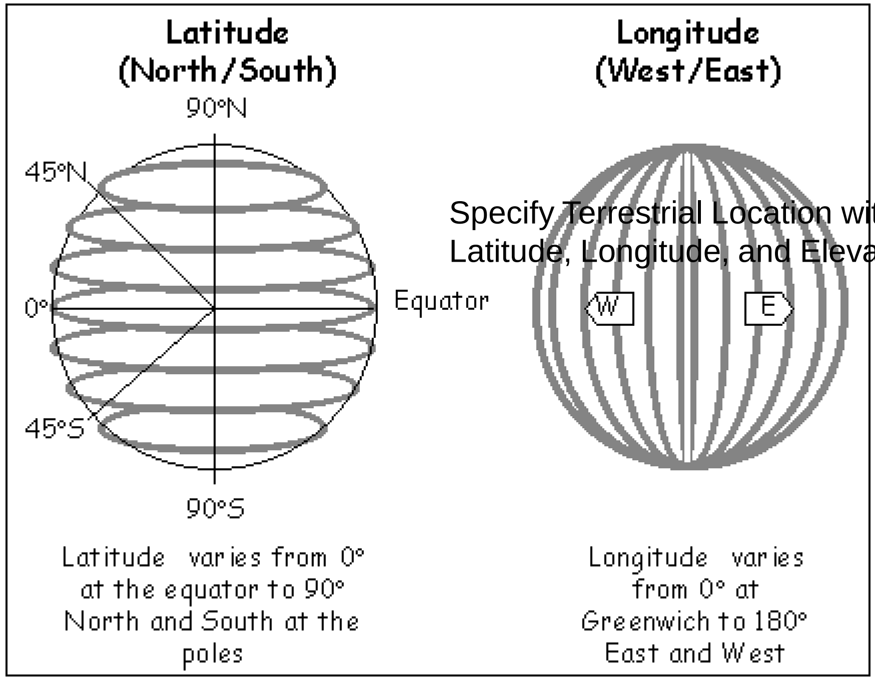


Fig. *The Tree of Life (Source: Berkeley)*

Geographic Coordinate System

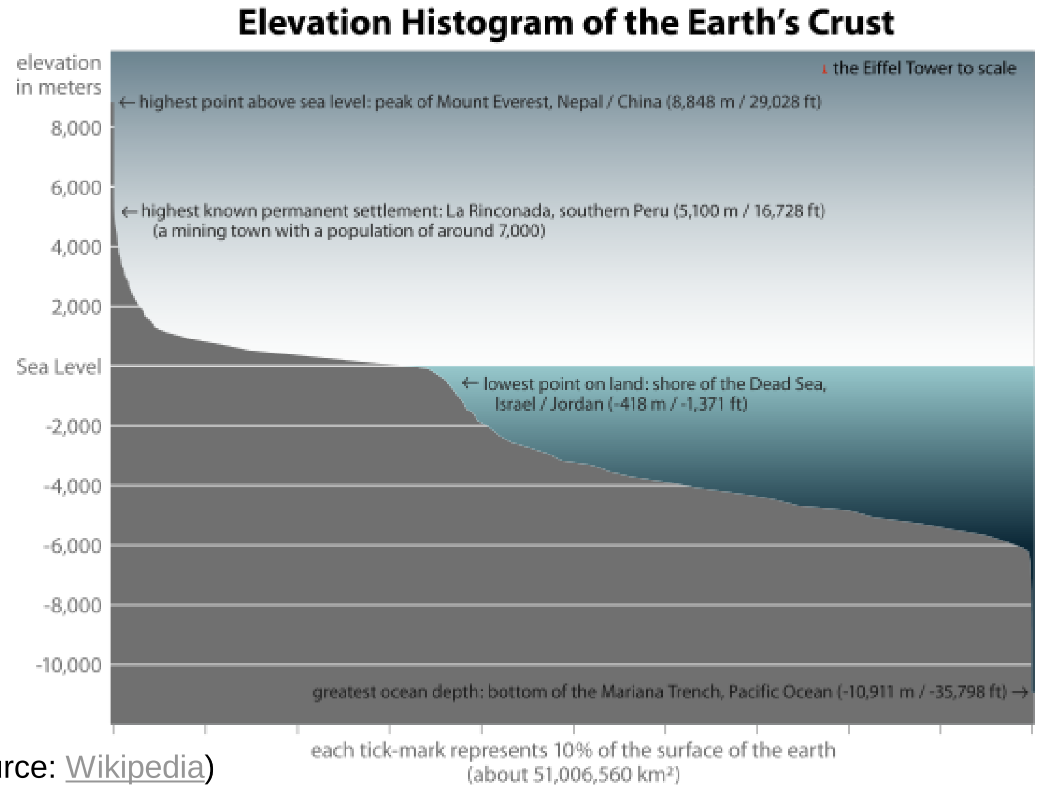
Specify Any Terrestrial Location using Latitude, Longitude, and Elevation.



Latitude (North/South)
90°N
45°N
0°
45°S
90°S
Equator
Latitude varies from 0° at the equator to 90° North and South at the poles
Longitude (West/East)
W
E
Longitude varies from 0° at Greenwich to 180° East and West

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.

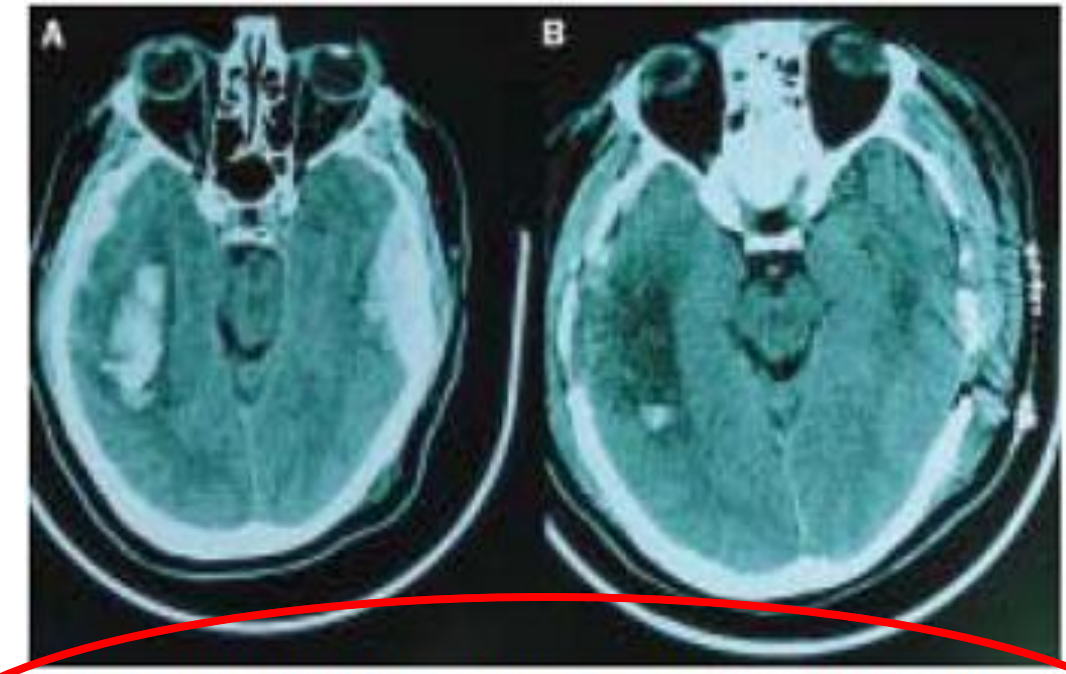


Figure 2: Computed tomography of a comatose patient with a left temporal epidural haematoma, right parenchymal temporal lobe haematoma, and a right convexity subdural haematoma before and after craniotomy and evacuation of haematomas

- 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

Information Exposure Faults

Formalizing information exposure faults would help researchers and practitioners identify them and avoid related failures.

To describe them, we developed:

- A general descriptive Model of Information Exposure
- One new BF class:
  - ✓ Information Exposure (IEX)

Information Exposure (IEX)

Information Exposure

- Information and data can be stored, transferred, and used by digital systems.
- Information exposure (information leaks) occurs when
  the system inadvertently reveals sensitive information inappropriately.

→ Exposure of sensitive information can be harmful on its own.
In addition, it could enable further attacks.

- Through information exposure the software may reveal:
  - login credentials
  - private keys
  - state and system data
  - personal data
  - financial data
  - health data
  - business data.

Information Exposure – Related Terms

- The terms “data” and “information” are often used interchangeably
  - *Data* is a set of values of qualitative or quantitative variables.
  - *Information* is any entity or form that provides the answer to a question of some kind or resolves uncertainty.

- To what extent data is informative to someone depends on how unexpected it is to that person.
  - Data has no meaning, while
  - Information has meaning.

- Information and data are on a continuum:
  - Bits in memory are data. Without external context (meaning), the bits might represent an integer, a memory address, a set of flags or other low-level information.
  - At a higher level, the integer could be someone’s age, the number of characters in a document, or a temperature.

Information Exposure – Related Terms

- In software, information is generated by processing data.
- We distinguish between information that is sensitive and information that is not
  - Certain kinds of information can be indirectly sensitive:
    when revealed can lead to harmful consequences.

→ If sensitive information at rest is properly encrypted (see BF ENC),
then information cannot be exposed, assuming a secure decryption key.

→ If information is communicated via a secure channel, it cannot be exposed either.

- Sensitive information includes:
  - ✓ Credentials
  - ✓ System data
  - ✓ State data
  - ✓ Cryptographic data
  - ✓ Digital documents
  - ✓ Personally identifiable data
  - ✓ Business data.

Information Exposure – Related Terms

- Credentials include: passwords, tokens, smart cards, digital certificates, and biometrics, such as fingerprints, hand configuration, retina, iris, and voice characteristics.
- System data could be: pathnames, configurations, logs, and Web usage.
- State data includes: operational data, such as SQL (Structured Query Language) table and column names, and server names.
- Cryptographic data is: hashes, keys, and keying material, such as cryptographic keys, initialization vectors, shared secrets, domain parameters, random seeds, salts, and nonces.

Information Exposure – Related Terms

- Personally identifiable data – could be used to distinguish people:
  - personally identifiable information (PII):
    e.g. SSN, driver's license number, and ID card number
  - personally identifiable financial information (PIFI) –
    e.g. financial account numbers with security codes/ access codes/ passwords
  - payment card information:
    e.g. cardholder name, expiration date, card verification value (CVV2 for Visa), card validation code, for MasterCard (CVC2), personal identification number (PIN) or PIN block, content of magnetic stripe.
  - protected health information:
    e.g. patient medical record or payment history.
- Business data:
  - e.g. intellectual property and trade secrets, operational and inventory data, and industry-specific data, in addition to customer and employee data.

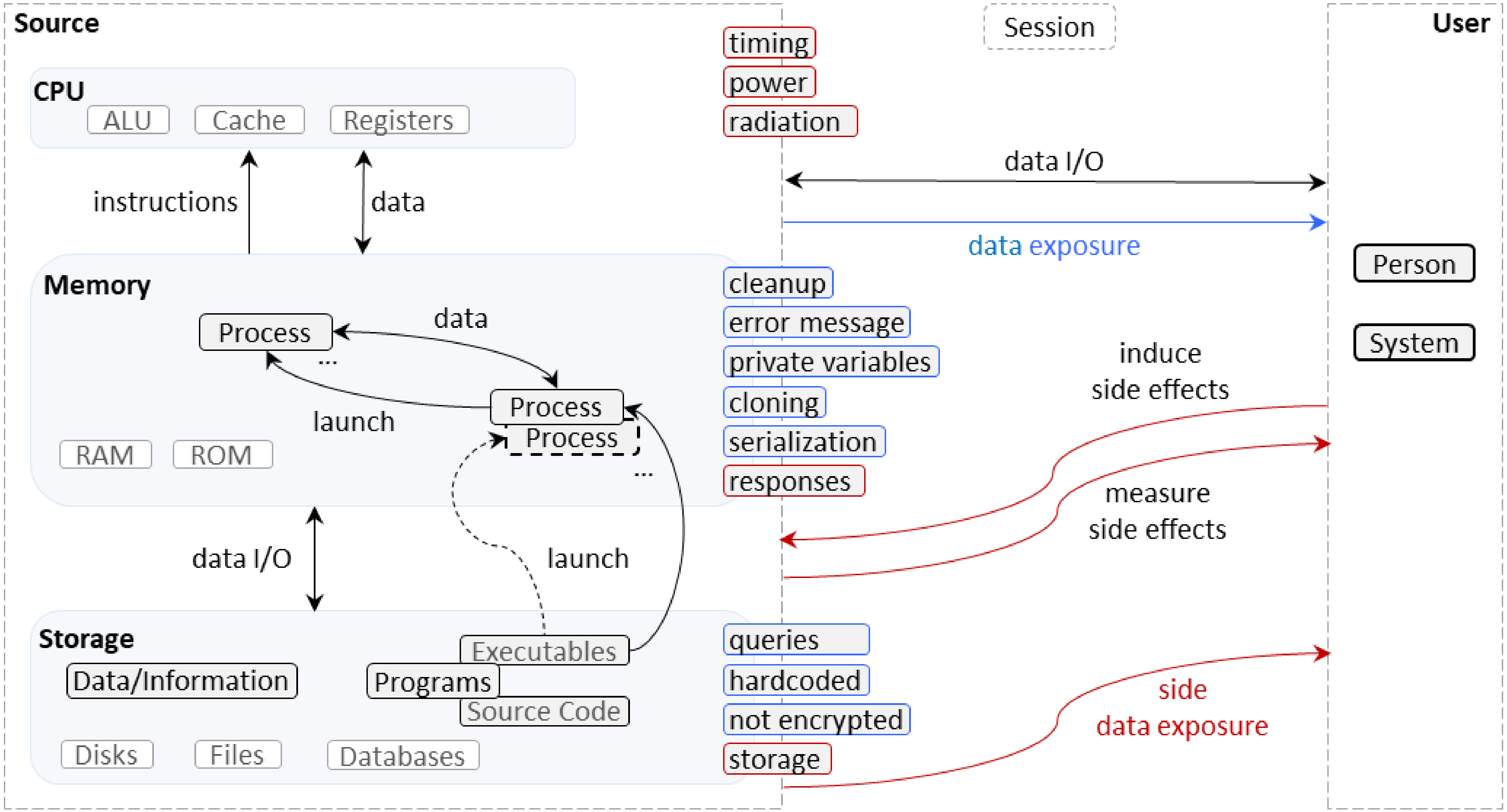
BF: Information Exposure Model

To understand information exposure,

- We developed a general descriptive model that
- Shows through what channels software could expose information.

→ *Exposure* is to any entity that should not have that information,
not just information that is a security concern.

BF: Information Exposure Model



BF: Information Exposure Model

- Information is stored on disks in files or in databases.
- Programs (source code and executables) are also stored on a disk in files and do not require any other resources.
  - A program is comprised of functions invocations.
  - Most functions process input data into output information or data.
  - A process is a program in execution and holds resources such as CPU, memory, storage, and I/O.
  - A program can involve more than one process.
  - A session is a temporary, interactive information interchange (I/O) between two or more devices, or between a computer and a user (e.g. login session).

BF: Information Exposure Model

- Information exposure may happen when:
  - either unintended information is carried, or
  - an unintended recipient gets the information.
- The exposure may be:
  - accidental or
  - because of intentional attacker actions.
- Exposure is through:
  - Legitimate Channels
    - incl. Diagnostic Channels
  - Side Channels
    - incl. Covert Channels

BF: IEX Model – Legitimate Channels

Information could leak through *legitimate channels* during normal use of software via:

- Information display
- Queries → e.g. query strings in SQL queries or GET requests
- Hardcoded information → e.g. passwords or cryptographic keys
- Class cloning
- Serializable classes
- Removing previously used information
- Buffer claeanup dead store removal
- Use of `realloc()`
- Session-ID length
- Sessions state boundaries
- Caching
- Session cleanup.

BF: IEX Model – Diagnostic Channels

- A *diagnostic channel* (error channel) is a legitimate channel that
  helps users and developers diagnose, find, and correct input or code errors.
- Information may be leaked via:
  - error messages
  - exception handling messages
  - other responses to erroneous inputs or erroneous data processing.

- If an attacker forces an internal fault,
  it may divulge sensitive information,
  including details on the software implementation logic.

BF: IEX Model – Side Channels

- A *side channel* is not intended to transmit information; however, it does transmit information.
- Information may be revealed or deduced due to discrepancies or behavioral inconsistencies:
  - conveying different responses (e.g. depending on if an operation is successful or not)
  - taking different time (e.g. CPU timing)
  - consuming different power
  - using different storage,
  - emitting different electromagnetic radiation.

- Behavioral inconsistency could be:
  - internal
  - external.

BF: IEX Model – Covert Channels

- A *covert channel* is a side channel that is
  created deliberately as a hidden communication channel.

→ A covert channel, for example, can be created by abusing TCP/IP.

- Unfortunately, a covert channel may be created by optimization techniques, such as:
  - compiler optimizations
  - speculative executions.

Examples of side/covert information exposure attacks are:

✓ Meltdown
✓ Spectre
✓ and the inference attacks.

BF IEX Model – More on Side Channels

- In a side channel, it is common for an attacker to control both:
  - the part that induces the side effect and
  - the part that measures it.
- In other cases, there could be two collaborating attackers:
  - an unauthorized user controlling the part that induces the side effect and
  - a third party controlling the part that measures it.
- There could be also:
  - only a passive attacker, who observes an existing (not induced) behavioral inconsistency.

→ Usually, statistical analysis of the measurements is involved.

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.

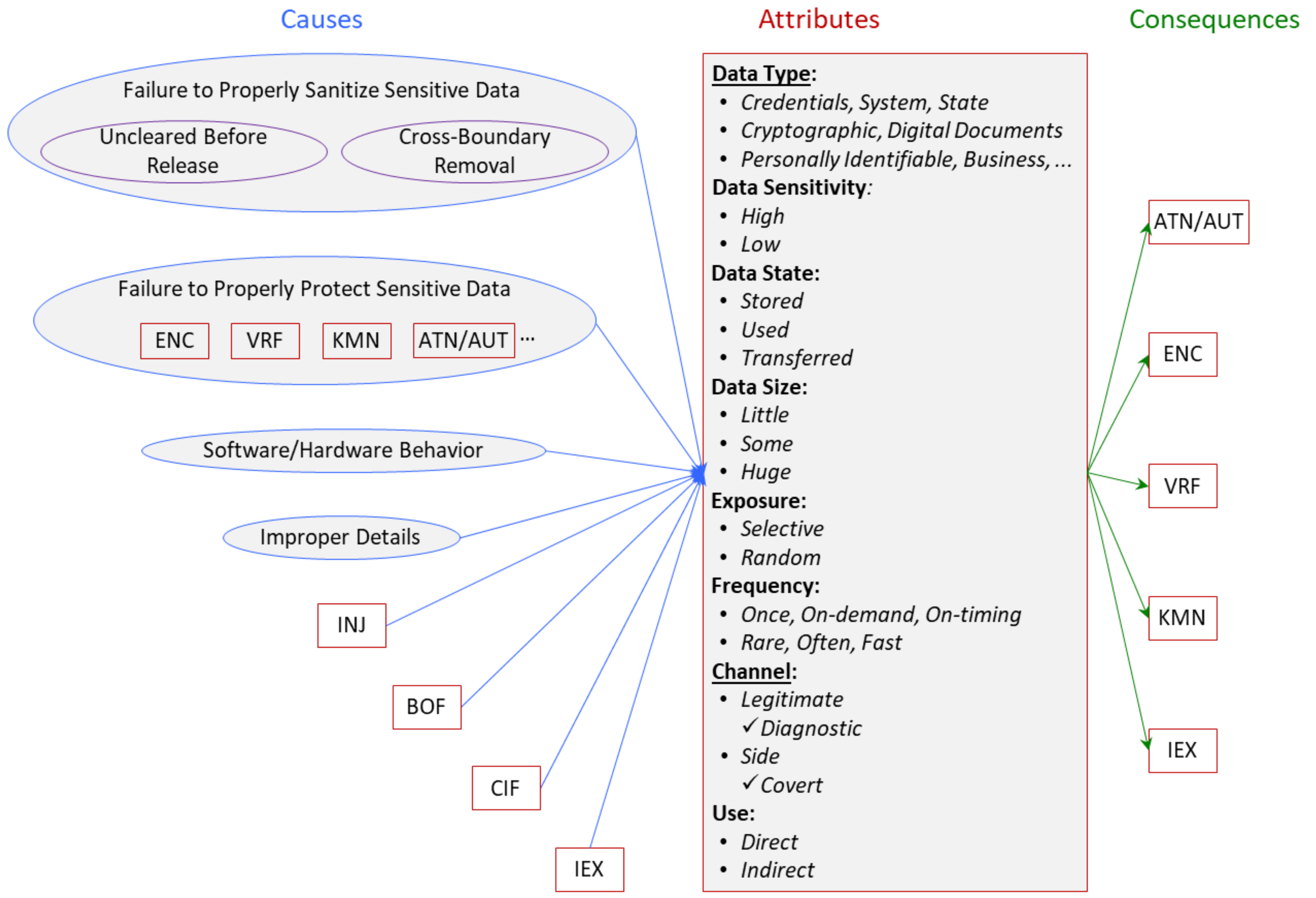
Related CWEs and SFPs:

- 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



Data Type:
- *Credentials, System, State*
- *Cryptographic, Digital Documents*
- *Personally Identifiable, Business, …*

Data Sensitivity:
- *High*
- *Low*

Data State:
- *Stored*
- *Used*
- *Transferred*

Data Size:
- *Little*
- *Some*
- *Huge*

Exposure:
- *Selective*
- *Random*

Frequency:
- *Once, On-demand, On-timing*
- *Rare, Often, Fast*

Channel:
- *Legitimate*
  - ✓ *Diagnostic*
- *Side*
  - ✓ *Covert*

Use:
- *Direct*
- *Indirect*

IEX: Attributes

- **Data Type** – Credentials, System Data, State Data, Cryptographic Data, Digital Documents, Personally Identifiable Data, Business Data, etc.

- **Data Sensitivity** – High, Low. This indicates the sensitivity level of leaked data/information.
  - Highly sensitive information that is properly encrypted,
    or information that is non-sensitive
    would not result in harm if exposed.
  - Non-sensitive information could be:
    - public records
    - phone books, or
    - online directories.

IEX: Attributes

- **Data State** – Stored, Used, Transferred. This reflects if the data is at rest, in use, or in transit.
  - Data can be at rest in:
    - files (e.g. initialization, include, temporary, configuration, log server, debug, cleanup, email attachment, login buffer, executable, backup, core dump, access control list, private data index)
    - directories (e.g. Web root, FTP root, CVS repository), or
    - on discs.
  - Data can be in use by functions/programs via:
    - source code (incl. comments)
    - threads
    - registries,
    - cookies
    - GUI
    - environmental variables.
  - Data can be also in transit:
    - between processes or
    - over a network.

IEX: Attributes

- **Data Size** – Little, Some, Huge.
  This indicates how much data/information is leaked.
  - These distinctions are important in some cases:
    → For example, Heartbleed might not have been a severe problem if it just exfiltrated a little data. The fact that it may exfiltrate a huge amount of data greatly increases the chance that very important information will be leaked.
- **Exposure** – Selective, Random.
  This reflects if an attacker can choose what information to expose or where.
  - Selective means the attacker can choose where and what to read.
  - Random is like going through the trash (e.g. Heartbleed).
- **Frequency** – Once, On-demand, On-timing, Rare, Often, Fast.
  This indicates how often the exposure can/does occur.
  - On-timing means depending on timing (e.g. in a race condition).

Note: `Frequency * Size = Rate.`

IEX: Attributes

- **Channel** – Legitimate, Diagnostic, Side, Covert.
  This indicates the medium by which information was leaked.

- **Use** – Direct, Indirect.
  - Direct means leaked data/information is valuable on its own.
  - Indirect means it is only useful for launching other attacks.

→ IEX is a high level class, so sites do not apply.

IEX: Causes and Consequences

In the graph of causes:

- *Uncleared Before Release* means information going from one control sphere back to the general pool.
- *Cross-Boundary Removal* means information going from one control sphere to another control sphere.
  (A control sphere is a set of resources and behaviors that are accessible to a single actor or a group of actors that all share the same security restrictions.)
- *Protect Sensitive Data* also covers preparing sensitive data.
- *Software/Hardware Behavior* covers algorithms and execution.
  Observable behavior (time, power, cache lines) depends on the data.
- *Improper Details* include: passwords, paths, SQL query structure/logic, etc. in error/exception, etc. messages.

IEX: Causes and Consequences

- ENC includes:
  - failure to encrypt (cleartext storage, recoverable format storage, cleartext transmission)
  - failure to properly encrypt (inadequate encryption strength, use of risky/broken cryptographic algorithm, missing required cryptographic step, use of hard-coded cryptographic key).
- ATN/AUT includes:
  - improper authentication
  - credentials compromise
  - account access.
- INJ includes:
  - adding commands and
  - masking legitimate commands or information.
- CIF includes limiting the number of failed log in attempts
  (if there is no limit, account names or passwords may be discovered by brute force attacks).

Note: One IEX fault may lead to another IEX.

IEX: Causes and Consequences

Note: One IEX fault may lead to another IEX.

→ e.g.,

IEX of all client credit cards may be caused by
earlier IEX of the password for a privileged account.

IEX: Example 1 – CVE-2007-5172

CVE-2007-5172:

“Quicksilver Forums before 1.4.1 allows remote attackers to obtain sensitive information by causing unspecified connection errors, which reveals the database password in the resulting error message.“ [1]

[1] The MITRE Corporation, CVE-2001-1141, http://cve.mitre.org/cgi-bin/cvename.cgi?name=cve-2007-5172.

IEX: Example 1 – CVE-2007-5172

CVE-2007-5172 description using IEX taxonomy:

IEX 1 of password leads to ATN leads to IEX 2.
IEX 1
Cause: Improper Details (error message displays password)
Attributes:
- Data Type: Credentials (password)
- Data Sensitivity: High
- Data State: Stored
- Data Size: Little
- Exposure: Selective
- Frequency: On-Demand
- Channel: Diagnostic (connection error message)
- Use: Indirect

Consequences: ATN.
ATN (to be described once the ATN class is developed).

IEX 2
Cause: Failure to Properly Protect Sensitive Data (password)
Attributes:
- Data Type: Any (user data)
- Data Sensitivity: Low/High
- Data State: Stored
- Data Size: Huge
- Exposure: Selective
- Frequency: On-Demand
- Channel: Legitimate
- Use: Direct (valuable on its own)

Consequences: Any IEX consequence.

IEX: Example 2 – CVE-2004-0243

CVE-2004-0243:

“AIX 4.3.3 through AIX 5.1, when direct remote login is disabled, displays a different message if the password is correct, which allows remote attackers to guess the password via brute force methods.“ [1]

[1] The MITRE Corporation, CVE-2001-1141, http://cve.mitre.org/cgi-bin/cvename.cgi?name=cve-CVE-2004-0243.

IEX: Example 2 – CVE-2004-0243

CVE-2004-0243 description using IEX taxonomy:

IEX 1 of password leads to ATN leads to IEX 2.
IEX 1
Cause: Program Behavior
(different responses for correct vs incorrect password)
Attributes:
- Data Type: Credentials (password)
- Data Sensitivity: High
- Data State: Used
- Data Size: Little
- Exposure: Selective
- Frequency: On-Demand
- Channel: Side (response inconsistency – message replies allow brute force password guessing)
- Use: Indirect

Consequences: ATN.
ATN (to be described once the ATN class is developed).

IEX 2
Cause: Failure to Properly Protect Sensitive Data (password)
Attributes:
- Data Type: Any (user data)
- Data Sensitivity: Low/High
- Data State: Stored
- Data Size: Huge
- Exposure: Selective
- Frequency: On-Demand
- Channel: Legitimate
- Use: Direct (valuable on its own)

Consequences: Any IEX consequence.

IEX: Example 3 – CVE-2017-5753 and CVE-2017-5715 (Spectre)

CVE-2017-5753:

“Systems with microprocessors utilizing speculative execution and branch prediction may allow unauthorized disclosure of information to an attacker with local user access via a side-channel analysis.“ [1]

CVE-2017-5715:

“Systems with microprocessors utilizing speculative execution and indirect branch prediction may allow unauthorized disclosure of information to an attacker with local user access via a side-channel analysis.“ [2]

[1] The MITRE Corporation, CVE-2001-1141, http://cve.mitre.org/cgi-bin/cvename.cgi?name=cve-CVE-2017-5753.
[2] The MITRE Corporation, CVE-2001-1141, http://cve.mitre.org/cgi-bin/cvename.cgi?name=cve-CVE-2017-5715.

IEX: Example 3 – CVE-2017-5753 and CVE-2017-5715 (Spectre)

CVE-2017-5753 and CVE-2017-5715 description using IEX taxonomy:

Cause: Hardware Behavior (CPU speculative execution)
Attributes:
- Data Type: Any (user's data)
- Data Sensitivity: High
- Data State: Stored
- Data Size: Huge
- Exposure: Selective
- Frequency: On-Demand
- Channel: Side (cache-based timing)
- Use: Any

Consequences: Any IEX consequence.

IEX: Example 4 – CVE-2017-5754 (Meltdown)

CVE-2017-5754:

“Systems with microprocessors utilizing speculative execution and indirect branch prediction may allow unauthorized disclosure of information to an attacker with local user access via a side-channel analysis of the data cache.“ [1]

[1] The MITRE Corporation, CVE-2017-5754, http://cve.mitre.org/cgi-bin/cvename.cgi?name=cve-CVE-2017-5754.

IEX: Example 4 – 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.

IEX – Conclusion

- We presented:
  - a general model of Information Exposure and
  - defined a new BF class, (IEX), including:
    - a rigorous definition
    - static attributes of the class, and
    - their related dynamic properties, such as proximate causes, consequences and sites.
- IEX is a very pervasive class, as:
  - many vulnerabilities lead to IEX, and
  - IEX may further lead to other faults.
- We analyzed particular vulnerabilities related to IEX and provide clear IEX descriptions.
- We showed that the BF-structured descriptions are quite concise,
  while still far clearer than unstructured explanations that we have found.

- We had to refine many of the previously defined BF classes.

IEX – Conclusion

- IEX joins other rigorously-defined BF classes, such as:

1. Randomness Cluster (RND)
   - Pseudo-Random Number Bugs (PRN)
   - True-Random Number Bugs (TRN)
2. Cryptography Cluster (CRY)
   - Key Management Bugs (KMN)
   - Encryption Bugs (ENC)
   - Verification Bugs (VRF)
3. Access Control Cluster (ACC)
   - Authentication Bugs (ATN)
   - Authorization Bugs (AUT)

IEX – Conclusion

- Lessons Learned and Future Work
  - At first, we had difficulties distinguishing strictly security-related leaks from other information exposure. However, we realized that “information exposure” is to any entity that should not have that information, not just leaks that are a security concern.
  - An IEX fault could actually be a cause of another IEX or the consequence of a preceding IEX. We learned that as we add more classes, such as IEX, we start chaining classes, consequences and causes in BF.
  - Work on explaining more information exposure faults using IEX and chains of BF classes will help us determine where our BF taxonomy needs refinement.
  - We will have to expand and refine many of the previously created BF descriptions of faults.

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

Randomness Cluster (RND):

- Pseudo-Random Number Bugs (PRN)
- True-Random Number Bugs (TRN)

RND: Randomness Classes in BF

Random number generators may have weaknesses (bugs) and applications using such may become vulnerable to attacks.

Formalization of randomness bugs would help researchers and practitioners identify them and avoid security failures.

We have developed a general descriptive model of randomness and have defined two Randomness BF classes:

- True-Random Number Bugs (TRN)
- Pseudo-Random Number Bugs (PRN)

Randomness

- Randomness has application in many fields, including:
  - ✓ Cryptography
  - ✓ Simulation
  - ✓ Statistics
  - ✓ Politics
  - ✓ Science
  - ✓ Gaming.
- Any specific use has its own requirements for randomness
  - e.g., random bit generation for cryptography or security purposes has stronger requirements than generation for other purposes.

    For cryptography or security purposes, NIST recommends use of cryptographically secure PRBGs: subject to the requirements in NIST SP 800-90A, NIST SP 800-90B and NIST SP 800-90C.
- Satisfying the requirements for a particular use can be surprisingly difficult.
- Weaknesses (bugs) in RNGs may lead to:
  - ✓ wrong results from the algorithms that use the generated numbers or
  - ✓ allow attackers to recover secret values, such as passwords and cryptographic keys.

Randomness Generation

We separate randomness generation in two distinct processes:

- True-random number generation:
  - ➢ nondeterministic (full entropy)
  - ➢ uses entropy sources.
- Pseudo-random number generation:
  - ➢ deterministic
  - ➢ uses true-random numbers as seeds.

  – PRBGs are used to extend the true-random seeds, produced by a TRBG –
  if the seed has length `n`, the output of the PRBG can have length `m`, where `m > n`.

  → However, a PRBG cannot increase the entropy of its seed.

  – It is possible though for a PRBG to use non-random seeds
  (e.g., for generating random numbers for simulation or game algorithms).

Randomness Attacks

Examples of attacks related to randomness generation are:

- Direct RSA common factor attack
- Cryptanalytic attack
- Input based attack
- State compromise attack.

BF: RND Model

Our BF randomness bugs (RND) model:

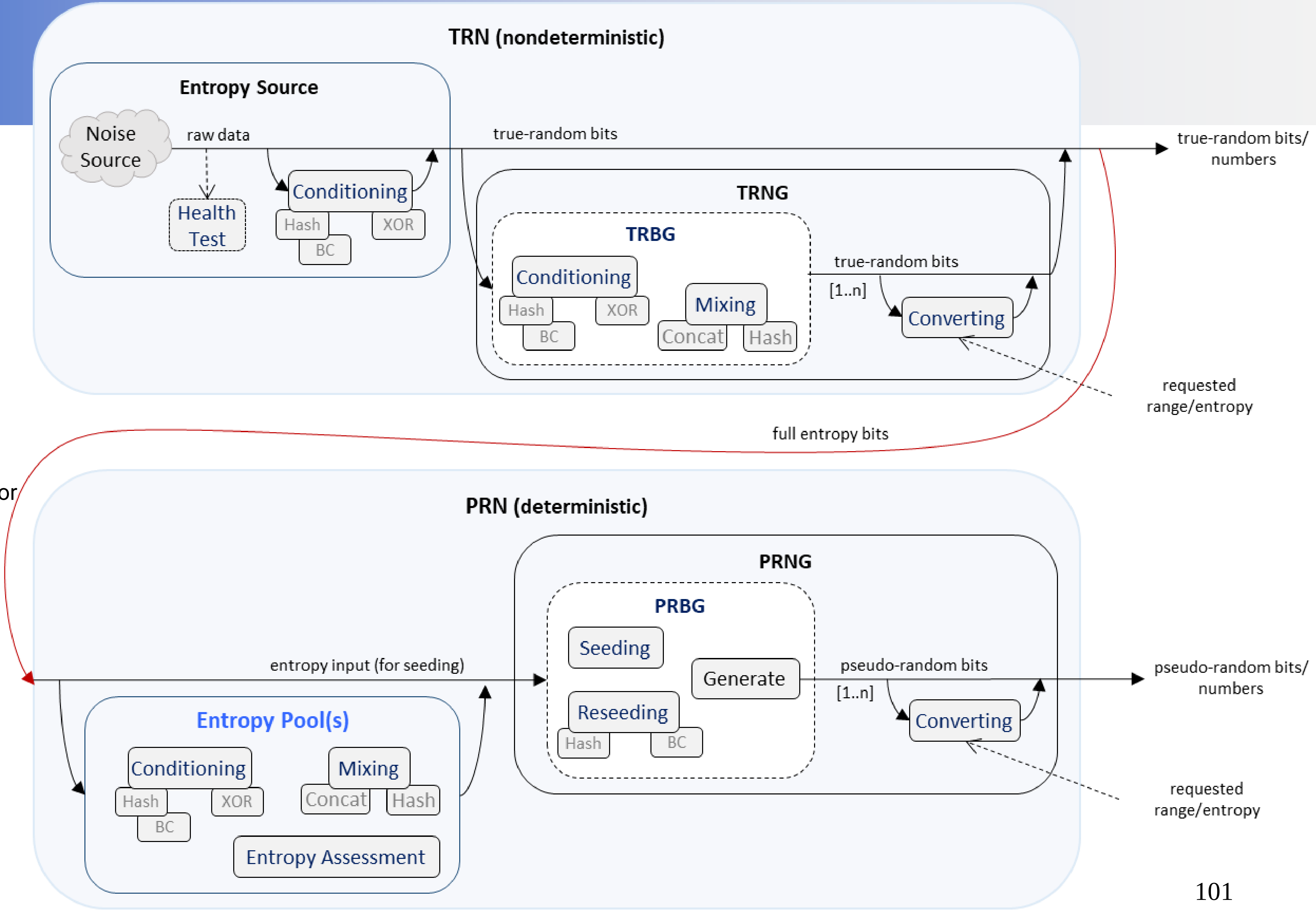
- Shows in which software components of TRNG and PRNG bugs can occur
- Is descriptive and not prescriptive
  → Should not be used as a model for construction of RBGs
  (🔒 NIST SP 800-90C specifies construction of RBGs using the mechanisms and entropy sources described in SP 800-90A and SP 800-90B, respectively.)

BF: RND Model

TRN – True-Random Number Bugs
PRN – Pseudo-Random Number Bugs

TRBG: True-Random Bit Generator
TRNG: – True-Random Number Generator

PRBG – Pseudo-Random Bit Generator
PRNG – Pseudo-Random Number Generator
BC – Block Cipher



BF: RND Model

Two distinct processes:

- TRN covers bugs related to:
  - entropy sources
  - TRBGs
  - TRNGs.
- PRN covers bugs related to:
  - entropy pools
  - PRBGs
  - PRNGs.

Although, output from the former process may be used as input to the latter (see the red arrow), they are distinct as bugs related to each have different causes, attributes, and consequences.

- The random bits are optionally converted in a pseudo-random number based on the range that applications provide as an argument.
- If live entropy source is used, the PRBG is said to support prediction resistance.
- PRNGs are algorithmic and can have bugs. Most PRNGs are not cryptographically secure.

BF: True-Random Number Bugs (TRN)

- We define True-Random Number Bugs (TRN) as:

  The software generated output does not satisfy all use-specific true-randomness requirements.

  Note that the output sequence is of random bits, where values are obtained from one or more sources of entropy.

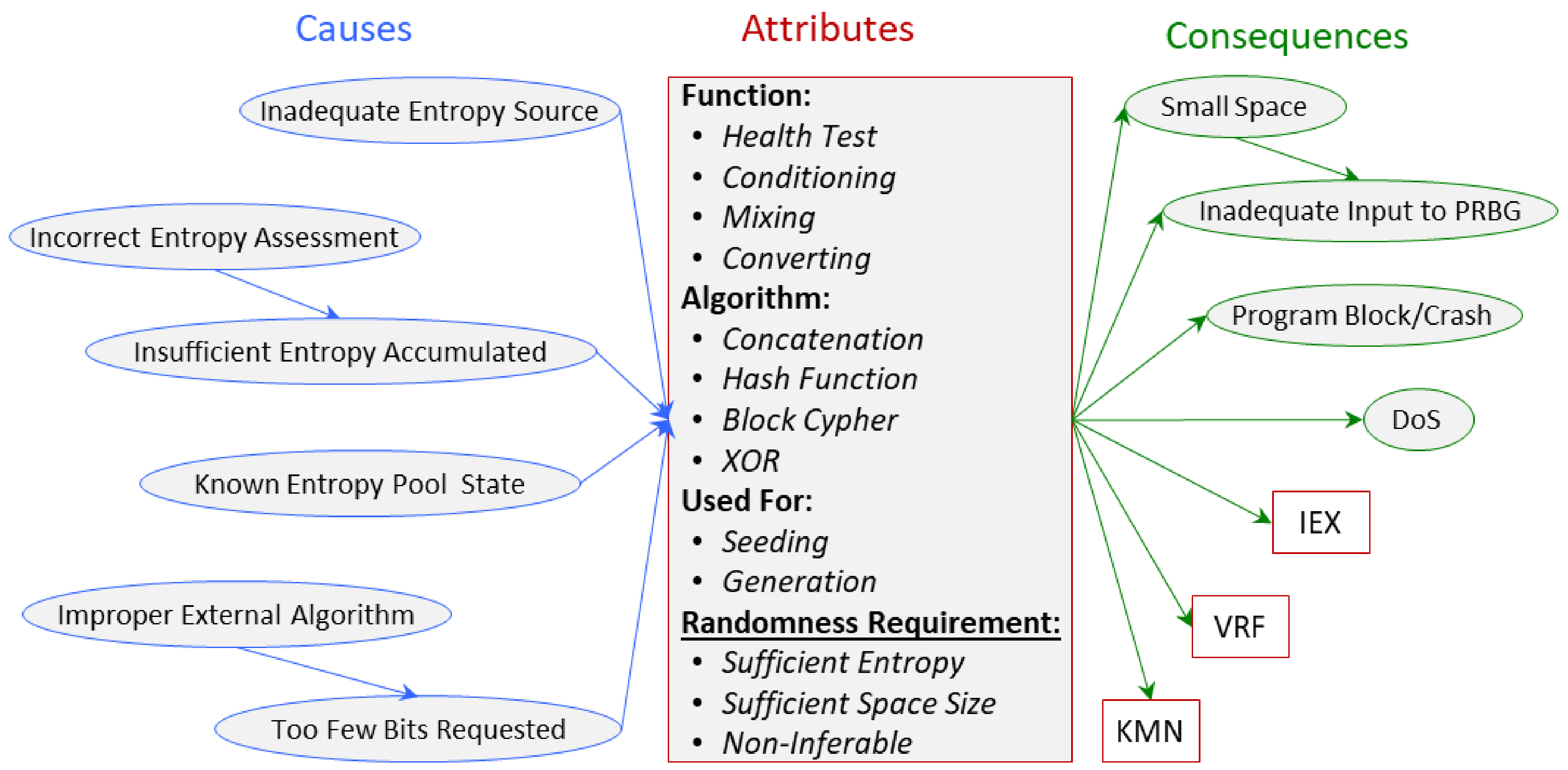
  TRN is related to: PRN, ENC, VRF, KMN, IEX.

Related CWEs, SFPs:

✓ SFP Primary Cluster: Predictability, which is CWE-905 with members CWE 330 to 344.

✓ Among them, TRN CWEs: 330, 331, 332, 333, 334, 337, 339, 340, 341, 342, 343.

TRN: Causes, Attributes, and Consequences



Function:
- *Health Test*
- *Conditioning*
- *Mixing*
- *Converting*

Algorithm:
- *Concatenation*
- *Hash Function*
- *Block Cypher*
- *XOR*

Used For:
- *Seeding*
- *Generation*

Randomness Requirement:
- *Sufficient Entropy*
- *Sufficient Space Size*
- *Non-Inferable*

TRN: Attributes

- **Function:**
  - ✓ Health Test
  - ✓ Conditioning
  - ✓ Mixing
  - ✓ Output
  - ✓ Converting.
- **Algorithm:**
  - ✓ Hash Function
  - ✓ Block Cipher
  - ✓ XOR, etc.
- **Used For** (what the output sequence is used for)**:**
  - ✓ Seeding
    – as a seed for a PRNG
  - ✓ Generation
    – of passwords or cryptographic keying material (keys, nonces).

TRN: Attributes

- **Randomness Requirement** (this is the failed requirement):
  - ✓ Sufficient Entropy
  - ✓ Sufficient Space Size
  - ✓ Non-Inferable.

➢ The notion of entropy used here is min-entropy, which is a measure of how difficult it is to guess the most likely entropy source output. (Let `X` be a random variable such that the set of its possible values is finite. Let `P` be the set of probabilities of `X` having those values. Then the min-entropy of `X` is $-\log_2 \max(P)$.)

➢ Space size – the number of elements of the space of possible outputs.
(If the number of different outputs is not sufficiently large, there is a vulnerability to a brute force attack.)

➢ Non-inferable – one cannot recover from known (guessed) information anything about the TRBG output. (TRBGs used for cryptography/security must satisfy the Non-Inferable randomness requirement.)

→ TRN is a high level class, so sites do not apply.

TRN: Causes & Consequences

- In the graph of causes:
  - Incorrect Entropy Assessment may result in TRN output having insufficient entropy.

    For example, a certain number of bits with full entropy may be needed, and because of Incorrect Entropy Assessment, the output may have insufficient entropy.

- In the graph of consequences:
  - Inadequate Input to PRNG could be:
    repeating, weak, insufficiently random, predictable, small space seed or other input.
  - A program may crash or block if it runs out of random number".
  - Denial of Service (DoS) can be a direct consequence.
  - KMN as in finding private keys from public keys using a common factor attack.
  - VRF as of using a predictable random number with a signature algorithm, such as DSA.
  - IEX could be of the exact value of a generated random number or a small range of values from which the generated random number is easy to figure out.

TRN: Example – CVE-2008-0141

CVE-2008-0141:
“actions.php in WebPortal CMS 0.6-beta generates predictable passwords containing only the time of day, which makes it easier for remote attackers to obtain access to any account via a lostpass action.“ [1]

[1] The MITRE Corporation, CVE Common Vulnerabilities and Exposures, CVE-2008-0141.

TRN: Example – CVE-2008-0141

CVE-2008-0141 → using TRN taxonomy

Cause: Inadequate Entropy Sources (current date/time and user name)
Attributes:
- Function: Mixing
- Algorithm: Concatenation
- Used for: Generation (of password)
- Randomness Requirement: Non-Inferable (time known from password reset time, name known from user register)

Consequences: IEX (of password), leading to ATN (Authentication Fault)

BF: Pseudo-Random Number Bugs (PRN)

- We define Pseudo-Random Number Bugs (PRN) as: :
  The software generated output does not satisfy all use-specific pseudo-randomness requirements.

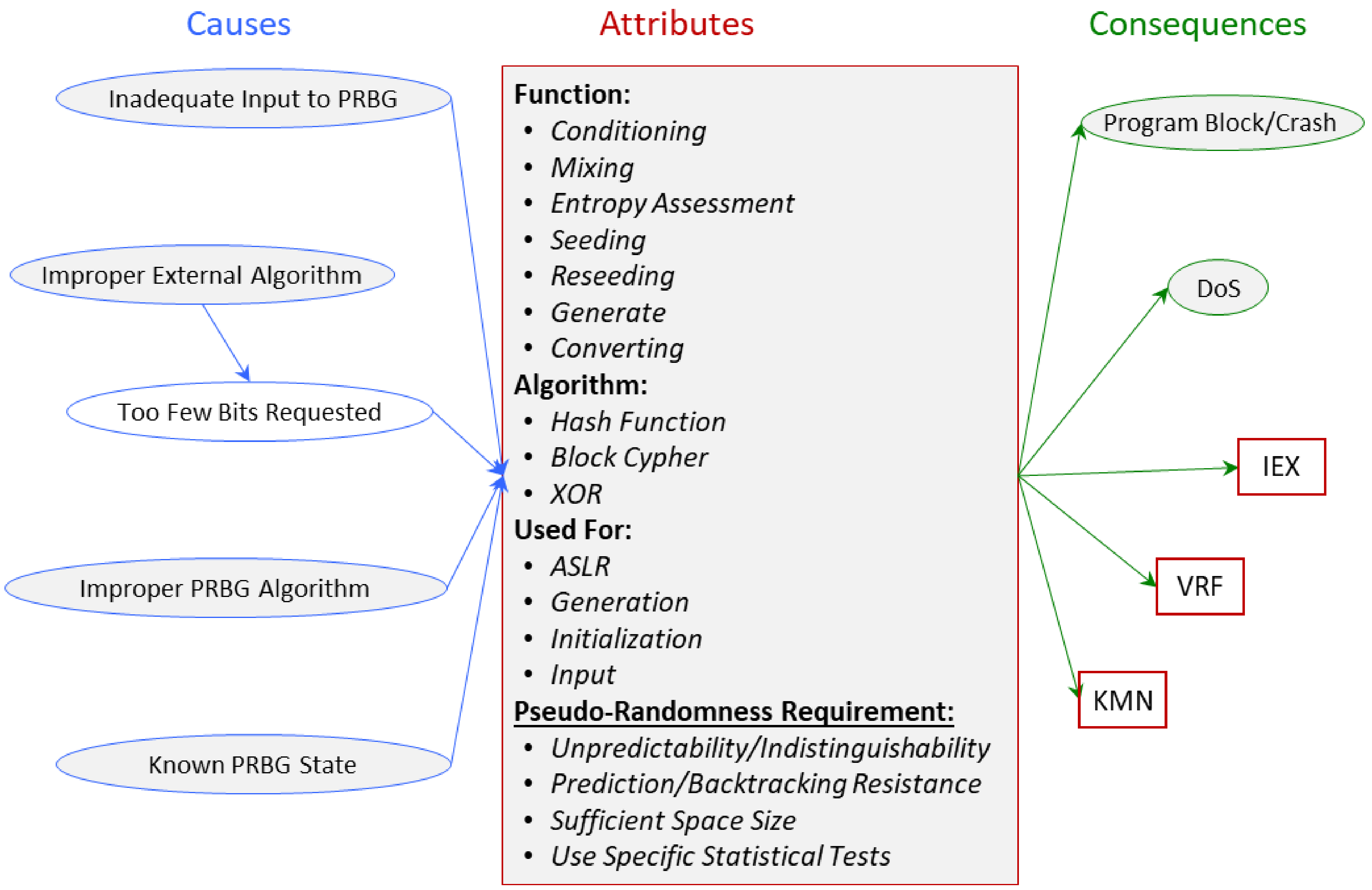
  Note that the output sequence is of random bits or numbers from a PRNG.

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

Related CWEs, SFPs:

- SFP Primary Cluster: Predictability, which is CWE-905 with members CWE 330 to 344.
- Among them, PRN CWEs: 330, 331, 332, 334, 335, 336, 337, 338, 339, 340, 341, 342, 343.

PRN: Causes, Attributes, and Consequences



Function:
- *Conditioning*
- *Mixing*
- *Entropy Assessment*
- *Seeding*
- *Reseeding*
- *Generate*
- *Converting*

Algorithm:
- *Hash Function*
- *Block Cypher*
- *XOR*

Used For:
- *ASLR*
- *Generation*
- *Initialization*
- *Input*

Pseudo-Randomness Requirement:
- *Unpredictability/Indistinguishability*
- *Prediction/Backtracking Resistance*
- *Sufficient Space Size*
- *Use Specific Statistical Tests*

PRN: Attributes

- **Function:**
  - ✓ Conditioning
  - ✓ Mixing
  - ✓ Entropy Assessment
  - ✓ Seeding
  - ✓ Reseeding
  - ✓ Generate
  - ✓ Converting.
- **Algorithm:**
  - ✓ Concatenation
    – the usual mixing of output from IID sources
  - ✓ Hash Function
  - ✓ Block Cipher
  - ✓ XOR.

PRN: Attributes

- **Used For** (what the output sequence is used for)**:**
  - ✓ ASLR (Address Space Layout Randomization)
  - ✓ Generation
    – of passwords or cryptographic keying material (keys, nonces)
  - ✓ Initialization
    – of cryptographic primitives (e.g., an initialization vector for cipher block chaining mode of encryption; or a salt for hashing)
  - ✓ Input to Algorithm
    – simulation, statistics, mathematics (e.g., Monte Carlo integration), or general algorithms.

PRN: Attributes

- **Pseudo-Randomness Requirement** (this is the failed requirement)**:**
  - ✓ Unpredictability/ Indistinguishability
  - ✓ Prediction/Backtracking Resistance
  - ✓ Sufficient Space Size
  - ✓ Use Specific Statistical Tests.

The pseudo-random output sequence should be statistically independent and unbiased. It should pass the use-specific statistical tests for randomness.

- ➢ Unpredictability – not possible to predict next generated output from previous output.
- ➢ Prediction Resistance –not possible to predict future output bits even if past or present state is known. (Prediction resistance is not possible without a live entropy source.)
- ➢ Backtracking Resistance – not possible to recover (backtrack) past output bits based on knowledge of the state at a given point in time.
- ➢ Indistinguishability for a PRNG – its output is computationally indistinguishable (i.e., by any probabilistic polynomial time algorithm) from a truly random sequence.
- ➢ Space size – the number of elements of the space of possible outputs.

PRN: Attributes

PRNGs used for cryptography/security must satisfy the following requirements:

- ✓ Unpredictability/Indistinguishability
- ✓ Backtracking Resistance
- ✓ Sufficient Space Size.

Prediction Resistance however is not always required – e.g. for PIV cards.

→ PRN is a high level class, so sites do not apply.

PRN: Example 1 – CVE-2001-1141

CVE-2001-1141:
“The Pseudo-Random Number Generator (PRNG) in SSLeay and OpenSSL before 0.9.6b allows attackers to use the output of small PRNG requests to determine the internal state information, which could be used by attackers to predict future pseudo-random numbers.“ [1]

[1] The MITRE Corporation, CVE-2001-1141, http://cve.mitre.org/cgi-bin/cvename.cgi?name=CVE-2001-1141.

PRN: Example 1 – CVE-2001-1141

CVE-2001-1141 description using PRN taxonomy:

Cause: Improper PRNG Algorithm (C md_rand – the secret PRNG state is updated with portion, as small as one byte, of the PRNG's previous output, which is not secret)
Attributes:
Function: Mixing (back into entropy pool)
Algorithm: Hash Function (SHA-1 – used for PRNG output and to update its internal secret state)
Used For: Generation (of cryptographic keying material – nonces, cryptographic keys)
Pseudo-Randomness Requirements: Sufficient Space Size and Unpredictability
(can be predicted from previous value through brute force)
Consequences: KMN>Generate with IEX of future keying material and ENC>IEX of sensitive data

PRN: Example 2 – CVE-2008-4107

CVE-2008-4107: “The (1) rand and (2) mt_rand functions in PHP 5.2.6 do not produce cryptographically strong random numbers, which allows attackers to leverage exposures in products that rely on these functions for security-relevant functionality, as demonstrated by the password-reset functionality in Joomla! 1.5.x and WordPress before 2.6.2, a different vulnerability than CVE-2008-2107, CVE-2008-2108, and CVE-2008-4102.” [1]

[1] The MITRE Corporation, CVE-2008-4107, http://cve.mitre.org/cgi-bin/cvename.cgi?name=CVE-2008-4107

PRN: Example 2 – CVE-2008-4107

CVE-2008-4107 description using PRN taxonomy:

Cause: Improper PRNG Algorithms (not cryptographically strong PHP 5 `rand` and `mt_rand`)
Attributes:
    Function: Generate (pseudo-random numbers)
    Algorithms: e.g., LCG or LFSR, Mersenne Twister
    Used For: Generation (of passwords)
    Pseudo-Randomness Requirements: Unpredictability/ Indistinguishability and Prediction Resistance
Consequence: IEX (of password), leading to ATN

PRN: Example 3 – CVE-2009-3238

CVE-2009-3238: “The get_random_int function in drivers/char/random.c in the Linux kernel before 2.6.30 produces insufficiently random numbers, which allows attackers to predict the return value, and possibly defeat protection mechanisms based on randomization, via vectors that leverage the function's tendency to "return the same value over and over again for long stretches of time.“ [1]

[1] The MITRE Corporation, CVE-2009-3238, http://cve.mitre.org/cgi-bin/cvename.cgi?name=CVE-2009-3238

PRN: Example 3 – CVE-2009-3238

CVE-2009-3238 description using TRN and PRN taxonomy:

An TRN leads to a PRN.

TRN

- **Cause:** Improper RNG Algorithm (same area is used for the hash array, allowing to repeatedly start from the same seed)
- **Attributes:**
  - Functions: Mixing (of pid and jiffies), Conditioning
  - Algorithms: Concatenation, Hash Function (MD4)
  - Used For: Seeding
- Randomness Requirements: Sufficient Entropy
- **Consequence:** Inadequate Input to PRNG (repeating seed)

PRN

- **Cause:** Inadequate Input to PRNG (repeating seed)
- **Attributes:**
  - Function: Generate (pseudo-random numbers)
  - Used For: ASLR
  - Pseudo-Randomness Requirement: Unpredictability
- **Consequences:** IEX of addresses

PRN: Example 4 – CVE-2017-15361 (ROCA – Return Of the Coppersmith Attack)

CVE-2017-15361:
“The Infineon RSA library 1.02.013 in Infineon Trusted Platform Module (TPM) firmware, such as versions before 0000000000000422 - 4.34, before 000000000000062b - 6.43, and before 0000000000008521 - 133.33, mishandles RSA key generation, which makes it easier for attackers to defeat various cryptographic protection mechanisms via targeted attacks, aka ROCA. Examples of affected technologies include BitLocker with TPM 1.2, YubiKey 4 (before 4.3.5) PGP key generation, and the Cached User Data encryption feature in Chrome OS.” [1]

[1] The MITRE Corporation, CVE-2017-15361, http://cve.mitre.org/cgi-bin/cvename.cgi?name=CVE-2017-15361

PRN: Example 4 – CVE-2017-15361 (ROCA)

CVE-2017-15361 description using KMN, and PRN taxonomy:

A KMN with inner PRN.

KMN:

Cause: Improper Algorithm Step (for generation of primes for RSA keys) leads to inner PRN

Attributes:

Cryptographic Data: Keying Material (keys)

Algorithm: RSA (key generation from two primes)

Operation: Generate (pair of public and private keys)

Consequences: Weak Public Key, which leads to IEX of Private Key.

Inner PRN:

Cause: Improper External Algorithm (generation of primes for RSA keys from random numbers and a constant related to keys size) leads to Too Few Bits Requested

Attributes:

Functions: Converting, Seeding (low entropy requested)

Used for: Generation (of secret prime numbers)

Randomness Requirement: Sufficient Space Size (e.g., one random number is only 37 bits for 512-bit RSA keys)

Consequence: IEX of generated primes (which format allows keys fingerprinting, factorization with Coppersmith algorithm, and finding random numbers and primes).

Cryptography Cluster (CRY):

- Key Management Bugs (KMN)
- Encryption Bugs (ENC)
- Verification Bugs (VRF)

Cryptography

- Broad, complex, and subtle area.
- Incorporates many clearly separate cryptographic processes, such as:
  - ✓ Encryption/ Decryption
  - ✓ Verification of data or source
  - ✓ Key management.
- Each cryptographic process
  → uses particular algorithms
  (e.g. symmetric/ asymmetric encryption, MAC, digital-signature)
  → to achieve particular security service.
  (e.g. confidentiality, integrity authentication, identity authentication, origin non-repudiation)

Cryptography Bugs

There are bugs if the software does not properly:

- Transform data into unintelligible form
  - ✓ Some transformations require keys – e.g. encryption and decryption
  - ✓ While others do not require keys – e.g. secret sharing.
- Verify:
  - ✓ Authenticity – data integrity, data source identity, origin for non-repudiation, content of secret sharing
  - ✓ Correctness – for uses such as zero-knowledge proofs.
- Manage keys
- Perform other operations.

Cryptography Attacks

Examples of attacks are:

- ✓ Spoofing messages
- ✓ Brute force attack
- ✓ Replaying instructions
- ✓ Timing attack
- ✓ Chosen plaintext attack (CPA)
- ✓ Chosen ciphertext attack (CCA)
- ✓ Exploiting use of weak or insecure keys
  (e.g. factorization of public key to obtain private key).

Cryptographic Store or Transfer

We use cryptographic store or transfer to illustrate the BF Cryptography Bugs (CRY) Classes:

- Encryption Bugs (ENC)
- Verification Bugs (VRF)
- Key Management Bugs (KMN)

Note: These classes may appear in many other situations such as:

- Self-sovereign identities
- Block ciphers
- Threshold cryptography.

We focus on transfer (or store) because it is:

✓ Well known a
✓ What most people think of when “cryptography” is mentioned.

Bugs in Cryptographic Store or Transfer

We define bugs in cryptographic store or transfer as:

> The software does not properly encrypt/decrypt, verify, or manage keys for data to be securely stored or transferred.

Bugs in Cryptographic Store or Transfer

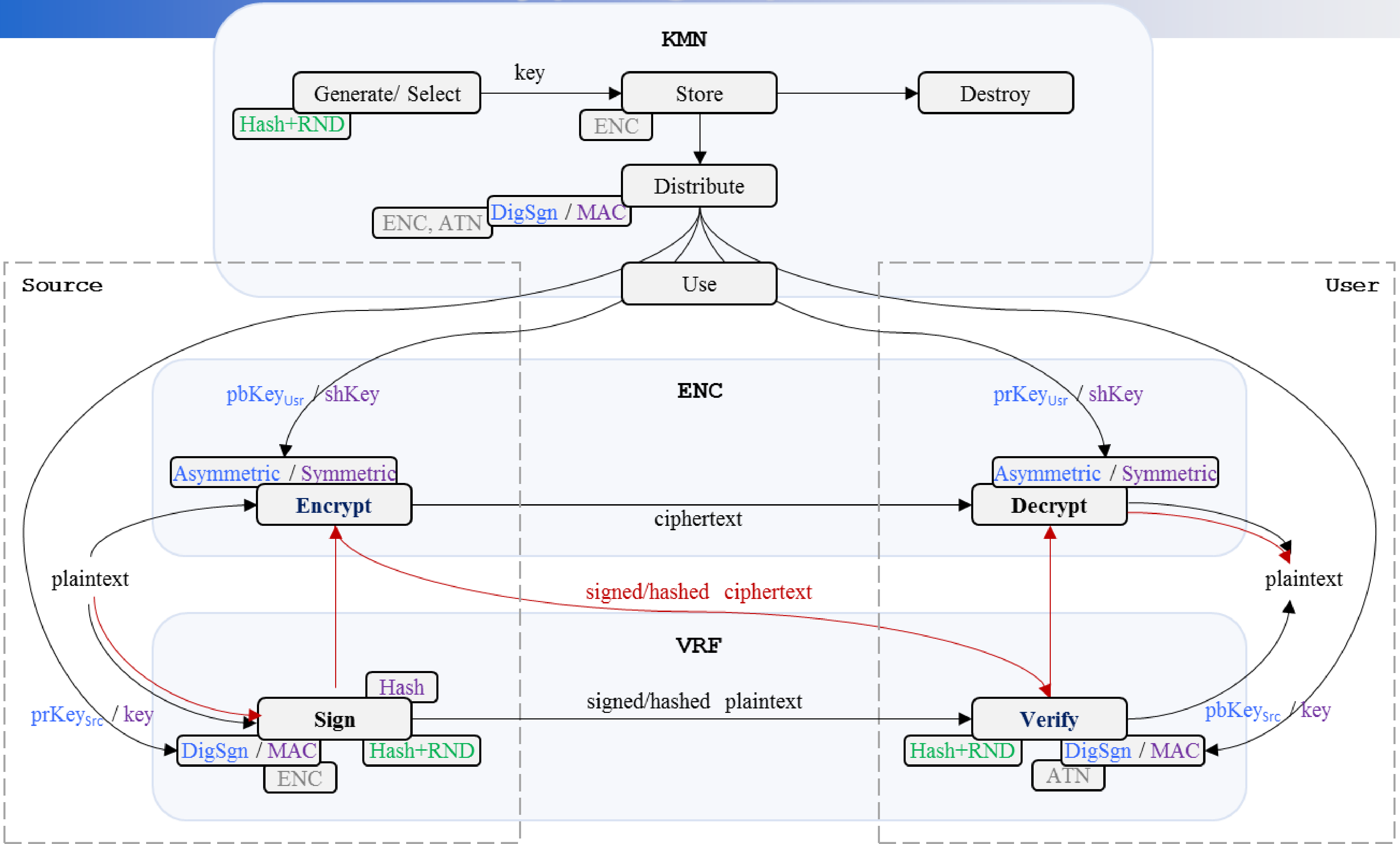
A modern, secure, flexible cryptographic storage or transfer protocol:

- Likely involves subtle interaction between following processes:
  - ✓ Encryption
  - ✓ Verification
  - ✓ Key Management
- May involve multiple stages of:
  - ✓ Agreeing on encryption algorithms
  - ✓ Establishing public and private keys
  - ✓ Creating session keys
  - ✓ Digitally signing texts for verification.

Thus, Encryption may use Key Management, which itself uses Encryption and Verification.

→ Need of a Model of these recursive interactions and
where potentially ENC, VRF, KMN, and other BF bugs could happen.

BF: Model of Cryptographic Store or Transfer



BF: CRY Model – Key Management Bugs

- KMN is a class of bugs related to key management.
- Key management comprises:
  - ✓ Key generation
  - ✓ Key selection
  - ✓ Key storage
  - ✓ Key retrieval and distribution
  - ✓ Determining and signaling when keys should be abandoned or replaced.

A particular protocol may use any or all of these operations.

BF: CRY Model – Key Management Bugs

- Key Management could be by:
  - ✓ a third party certificate authority (CA) – distributes public keys in signed certificates
  - ✓ the source
  - ✓ the user

Thus the Key Management area intersects the Source and User areas.

Key Management often uses a recursive round of encryption and decryption, and verification to establish a shared secret key or session key before the actual plaintext is handled.

BF: CRY Model – Encryption Bugs

- ENC is a class of bugs related to encryption.
- Encryption comprises:
  - ✓ Encryption by the source
  - ✓ Decryption by the user.

- Encryption/ decryption algorithms may be:
  - ✓ Symmetric – uses same key for both
  - ✓ Asymmetric – uses pairs of keys: one to encrypt, other to decrypt.

Public key cryptosystems are asymmetric.

The ciphertext may be sent directly to the user, and verification accompanies it separately.
The red line is a case where plaintext is signed or hashed and then encrypted.

BF: CRY Model – Verification Bugs

- VRF is a class of bugs related to verification.
- Verification:
  - Takes a key and either the plaintext or the ciphertext
    signs or hashes it then passes the result to the user.
  - User uses the same key or the other member of the key pair to verify source.

BF: CRY Model – Keys Usage

- Symmetric encryption – one secretly shared key (shKey) is used:
  - ✓ Source encrypts with shKey
  - ✓ User decrypts with shKey, too.

- Asymmetric encryption – pairs of mathematically related keys are used, source pair: (pbKey$_{Src}$, prKey$_{Src}$), user pair: (pbKey$_{Usr}$ and prKey$_{Usr}$):
  - ✓ Source:
    - ○ encrypts with pbKey$_{Usr}$
    - ○ signs with prKey$_{Src}$
  - ✓ User:
    - ○ decrypts with prKey$_{Usr}$
    - ○ verifies with pbKey$_{Src}$.

BF: Encryption Bugs (ENC)

- We define Encryption Bugs (ENC) as:

  The software does not properly transform sensitive data (plaintext) into unintelligible form (ciphertext) using cryptographic algorithm and key(s).
- We define also the Decryption Bugs as:

  The software does not properly transform ciphertext into plaintext using cryptographic algorithm and key(s).

Note that "transform" is for confidentiality.

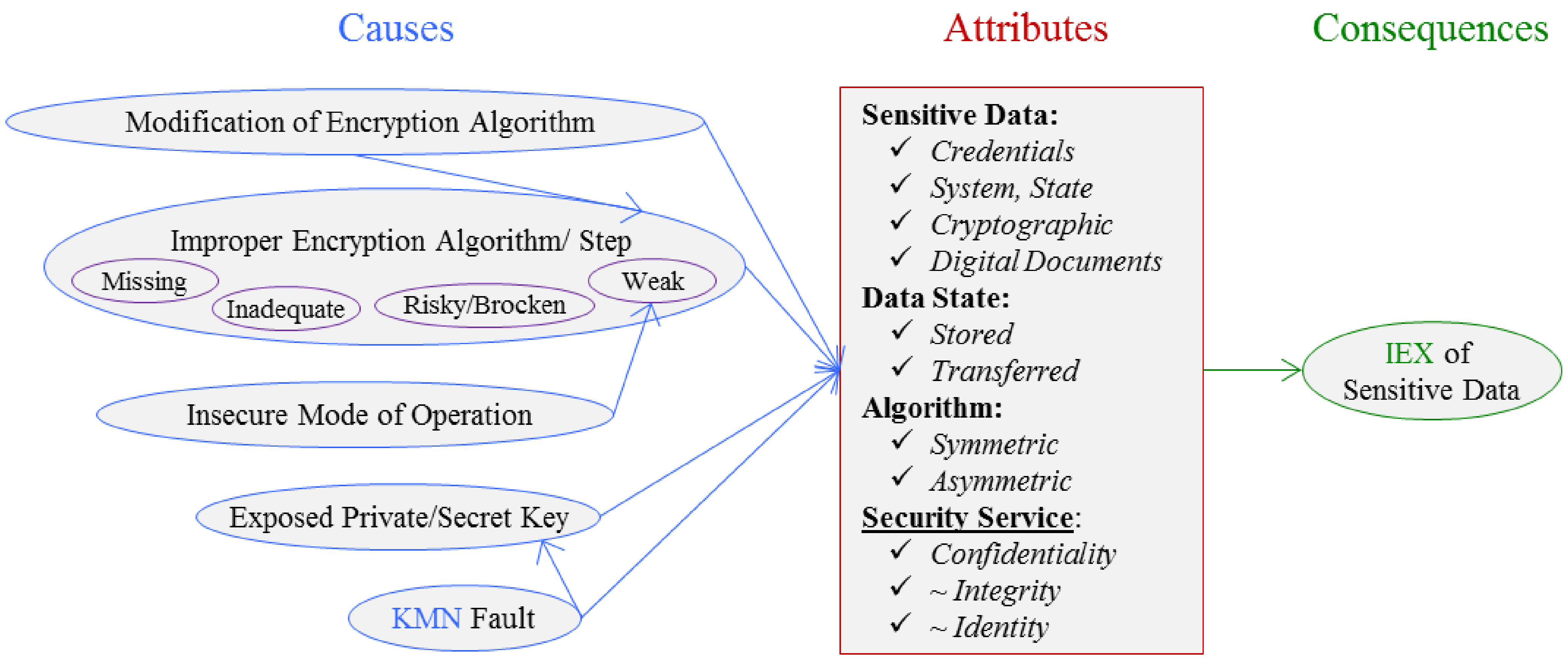
ENC is related to KMN, TRN, PRN, and IEX.

Related CWEs, SFPs and ST:

✓ CWEs: 256, 257, 261, 311-318, 325, 326, 327, 329, 780.

✓ SFP clusters: SPF 17.1 Broken Cryptography and SFP 17.2 Weak Cryptography under Primary Cluster: Cryptography.

ENC: Causes, Attributes, and Consequences



Causes
Attributes
Consequences
Modification of Encryption Algorithm
Improper Encryption Algorithm/ Step
Missing
Inadequate
Risky/Brocken
Weak
Insecure Mode of Operation
Exposed Private/Secret Key
KMN Fault
IEX of Sensitive Data

Sensitive Data:
- ✓ *Credentials*
- ✓ *System, State*
- ✓ *Cryptographic*
- ✓ *Digital Documents*

Data State:
- ✓ *Stored*
- ✓ *Transferred*

Algorithm:
- ✓ *Symmetric*
- ✓ *Asymmetric*

Security Service:
- ✓ *Confidentiality*
- ✓ *~ Integrity*
- ✓ *~ Identity*

ENC: Attributes

- **Sensitive Data** – This is secret (confidential) data.
  - ✓ Credentials: Password, Token, Smart Card, Digital Certificate, Biometrics (fingerprint, hand configuration, retina, iris, voice.)
  - ✓ System Data: Configurations, Logs, Web usage, etc.
  - ✓ State Data
  - ✓ Cryptographic Data: hashes, keys, and other keying material
  - ✓ Digital Documents.
- **Data State –** This reflects if data is in rest or use, or if data is in transit.
  - ✓ Stored: data in rest or use from files (e.g. ini, temp, configuration, log server, debug, cleanup, email attachment, login buffer, executable, backup, core dump, access control list, private data index), directories (Web root, FTP root, CVS repository), registry, cookies, source code & comments, GUI, environmental variables.
  - ✓ Transferred: data in transit between processes or over a network.

ENC: Attributes

- **Algorithm** –the key encryption scheme used to securely store/transfer sensitive data.
  - ✓ Symmetric (secret) key algorithms (e.g. Serpent, Blowfish) use one shared key.
  - ✓ Asymmetric (public) key algorithms (e.g. Diffie-Hellman, RSA) use two keys (public, private).
- **Security Service(s)** – that was failed by the encryption process
  - ✓ Confidentiality – the main security service provided by encryption.
  - ✓ ~Integrity, ~Identity Authentication – in some specific modes of encryption.

→ ENC is a high level class, so sites do not apply.

ENC: Example 1 – CVE-2002-1946

CVE-2002-1946 → using ENC taxonomy

Cause: Weak Encryption Algorithm (one-to-one mapping)

Attributes:

Sensitive Data: Credentials (passwords)

Data State: Stored (in registry)

Algorithm: Symmetric (that allows obtaining shared key and decryption)

Security Service: Confidentiality

Consequence: IEX of Sensitive Data (credentials)

____

CVE-2002-1946: "Videsh Sanchar Nigam Limited (VSNL) Integrated Dialer Software 1.2.000, when the "Save Password" option is used, stores the password with a weak encryption scheme (one-to-one mapping) in a registry key, which allows local users to obtain and decrypt the password." [1]

[1] The MITRE Corporation, CVE Common Vulnerabilities and Exposures, CVE-2002-1946.

ENC: Example 2 – CVE-2002-1697

CVE-2002-1697 → using **ENC** taxonomy

- **Causes:** Insecure Mode of Operation (ECB) leads to Weak Encryption Algorithm (for same shared key produces same ciphertext from same plaintext)
- **Attributes:**
  - Sensitive Data: Any (Credentials, Cryptographic, …)
  - Data State: Transferred (over network)
  - Algorithm: Symmetric (that allows identifying patterns and data recovery)
  - Security Service: Confidentiality
- **Consequence:** IEX of Sensitive Data

CVE-2002-1697: "Electronic Code Book (ECB) mode in VTun 2.0 through 2.5 uses a weak encryption algorithm that produces the same ciphertext from the same plaintext blocks, which could allow remote attackers to gain sensitive information." [1]

[1] The MITRE Corporation, CVE-2002-1697, http://cve.mitre.org/cgi-bin/cvename.cgi?name=CVE-2002-1697.

BF: Verification Bugs (VRF)

- Our Definition:
  The software does not properly sign data, check and prove source, or assure data is not altered.

  Note that "check" is for identity authentication, "prove" is for origin (signer) non-repudiation, and "not altered" is for integrity authentication.

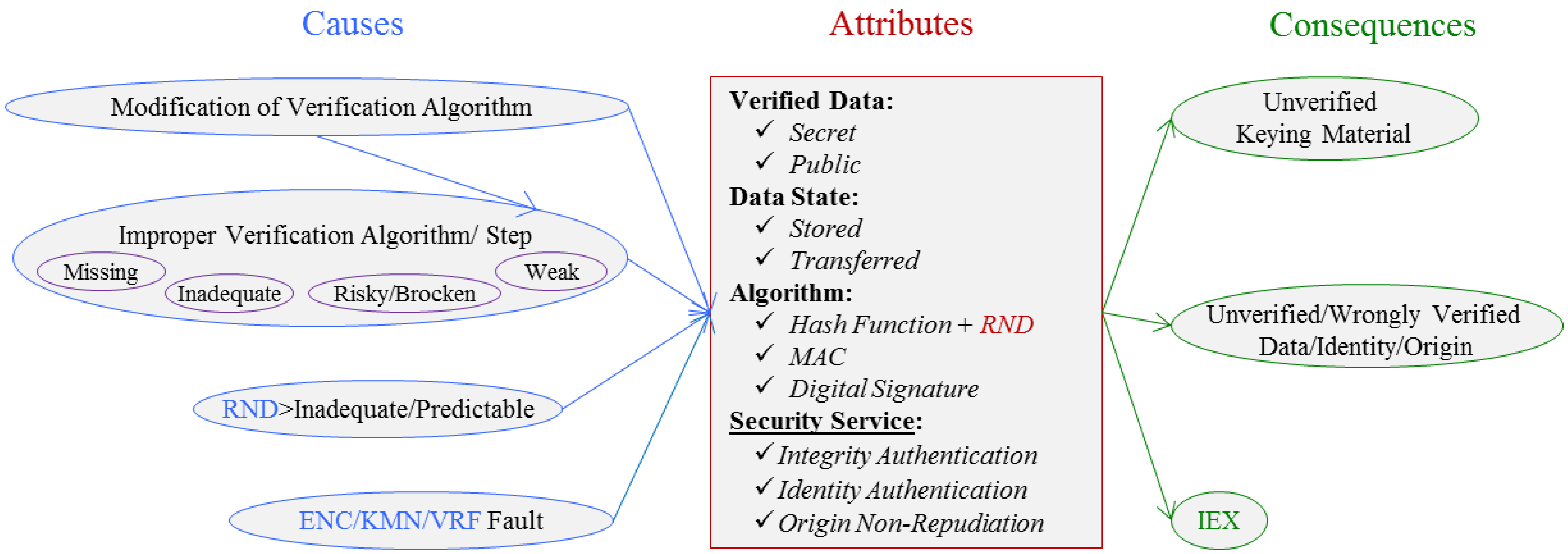
  VRF is related to KMN, TRN, PRN, ENC, ATN, IEX.

Related CWEs, SFPs and ST:

✓ CWEs: 295, 296, 347.

✓ SFP cluster: SFP 17.2 Weak Cryptography under Primary Cluster: Cryptography.

VRF: Causes, Attributes, and Consequences


Causes
Attributes
Consequences
Modification of Verification Algorithm
Improper Verification Algorithm/ Step
Missing
Inadequate
Risky/Brocken
Weak
RND>Inadequate/Predictable
ENC/KMN/VRF Fault
Unverified Keying Material
Unverified/Wrongly Verified Data/Identity/Origin
IEX

Verified Data:
- ✓ *Secret*
- ✓ *Public*

Data State:
- ✓ *Stored*
- ✓ *Transferred*

Algorithm:
- ✓ *Hash Function + RND*
- ✓ *MAC*
- ✓ *Digital Signature*

Security Service:
- ✓ *Integrity Authentication*
- ✓ *Identity Authentication*
- ✓ *Origin Non-Repudiation*

VRF: Attributes

- **Verified Data** – This is the data that needs verification. It may be confidential or public.
  - Secret (confidential) Data: cryptographic hashes, secret keys, or keying material.
  - Public Data: signed contract, documents, or public keys.
- **Data State –** This reflects if data is in rest or use, or if data is in transit.
- **Algorithm** – Hash Function + RND, Message Authentication Code (MAC), Digital Signature.
  - Hash functions are used for integrity authentication. They use RND.
  - MAC are symmetric key algorithms (one secret key per source/user), used for integrity authentication, identity authentication. It needs authentication code generation, source signs data, user gets tag for key and data, and verifies data by tag and key.
  - Digital Signature is an asymmetric key algorithm (two keys), used for integrity and identity authentication, and origin (signer) non-repudiation. It needs key generation, signature generation, and signature verification.

  MAC and Digital Signature use KMN and recursively VRF.

VRF: Attributes

- **Security Service** – This is the security service the verification process failed.
  - Data Integrity Authentication – for data and keys
  - Identity Authentication – for source authentication
  - Origin (Signer) Non-Repudiation – for source authentication.

→ VRF is a high level class, so sites do not apply.

VRF: Example 1 – CVE-2001-1585

CVE-2001-1585: “SSH protocol 2 (aka SSH-2) public key authentication in the development snapshot of OpenSSH 2.3.1, available from 2001-01-18 through 2001-02-08, does not perform a challenge-response step to ensure that the client has the proper private key, which allows remote attackers to bypass authentication as other users by supplying a public key from that user's authorized_keys file.“ [1]

[1] The MITRE Corporation, CVE Common Vulnerabilities and Exposures, CVE- 2001-1585.

VRF: Example 1 – CVE-2001-1585

CVE-2001-1585 description using VRF taxonomy:

CVE 2001-1585 → VRF

Cause Missing Verification Step (challenge-response) in public key authentication

Attributes:

- Verified Data: Any (Secret/ Public)
- Data State: Transferred (over network)
- Algorithm: Digital Signature (not using such allows private key not to be verified by public key)
- Security Service: Identity Authentication

Consequence: IEX

VRF: Example 2 – CVE-CVE-2015-2141

CVE-2015-2141: “The InvertibleRWFunction::CalculateInverse function in rw.cpp in libcrypt++ 5.6.2 does not properly blind private key operations for the Rabin-Williams digital signature algorithm, which allows remote attackers to obtain private keys via a timing attack. .” [1]

[1] The MITRE Corporation, CVE-2141, http://www.cve.mitre.org/cgi-bin/cvename.cgi?name=2015-2141

VRF: Example 2 – CVE-CVE-2015-2141

CVE-2015-2141 description using VRF taxonomy:

CVE 2015-2141 → VRF

Cause: Modification of Verification Algorithm by adding a step (blinding)

Attributes:

- Verified Data: Any (Secret/ Public)
- Data State: Transferred (over network)
- Algorithm: Digital Signature (Rabin-Williams) (that allows obtaining the private key in cases of incorrect unblinding)
- Security Service: Identity Authentication

Consequence: IEX

BF: Key Management Bugs (KMN)

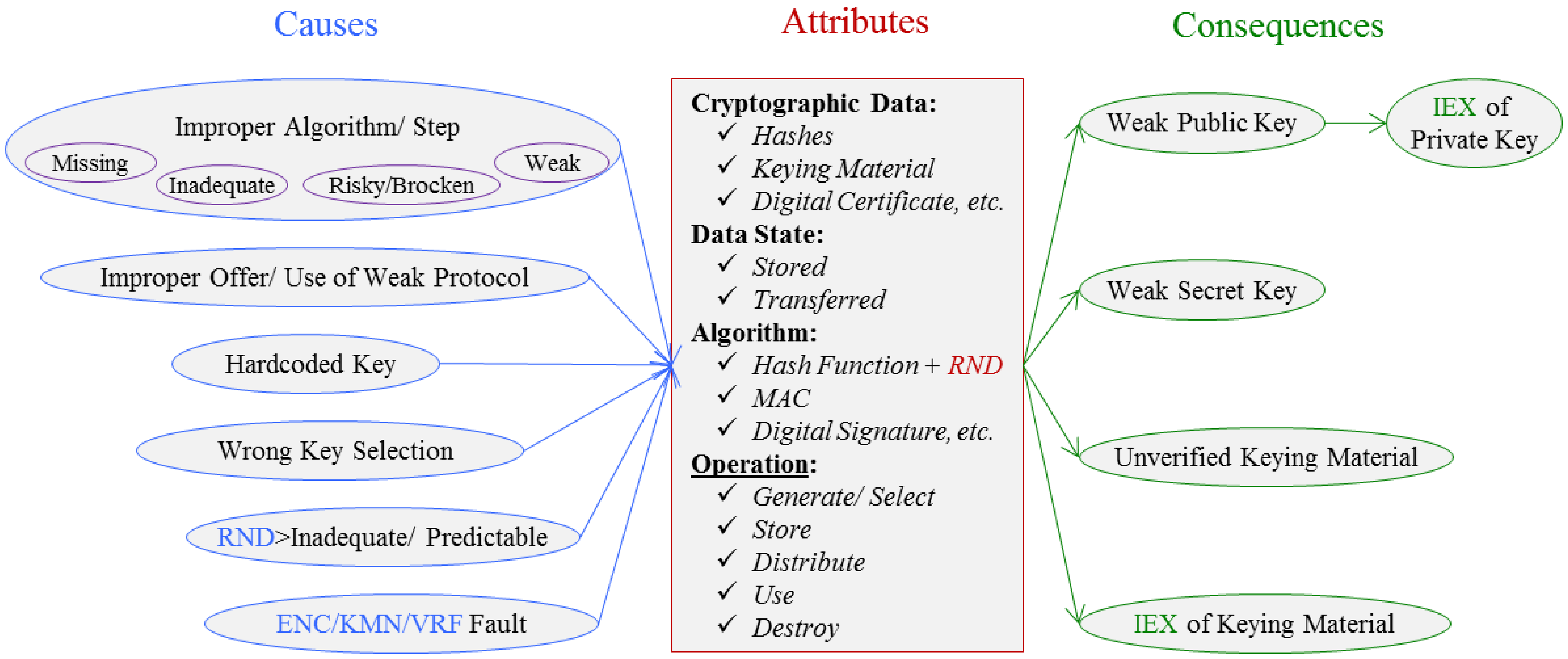
- Our Definition:
  The software does not properly generate, store, distribute, use, or destroy cryptographic keys and other keying material.

  KMN is related to ENC, TRN, PRN, VRF, IEX.

Related CWEs, SFPs and ST:

- ✓ CWEs: 321, 322, 323, 324.
- ✓ SFP clusters: SFP 17.2 Weak Cryptography under Primary Cluster: Cryptography and SFP 4.13 Digital Certificate under Primary Cluster: Authentication .

KMN: Causes, Attributes, and Consequences



KMN: Causes, Attributes, and Consequences

- **Cryptographic Data** – Hashes, Keying Material, Digital Certificate.
- **Data State –** This reflects if data is in rest or use, or if data is in transit.
- **Algorithm** – Hash Function + RND, MAC, Digital Signature.
- **<u>Operation</u>** –This is the failed operation: Generate uses RND.
  - Store – includes update and recover
  - Distribute – includes key establishment, transport, agreement, wrapping, encapsulation, derivation, confirmation, shared secret creation; uses ENC and KMN (reclusively)
  - Use
  - Destroy.

→ KMN is a high level class, so sites do not apply.

BF: KMN Example (FREAK)– CVE-2015-0204, CVE-2015-1637, CVE-2015-1067

FREAK – Factoring attack on RSA-ExportKeys

CVE-2015-0204: “The ssl3_get_key_exchange function in s3_clnt.c in OpenSSL before 0.9.8zd, 1.0.0 before 1.0.0p, and 1.0.1 before 1.0.1k allows remote SSL servers to conduct RSA-to-EXPORT_RSA downgrade attacks and facilitate brute-force decryption by offering a weak ephemeral RSA key in a noncompliant role, related to the "FREAK" issue. NOTE: the scope of this CVE is only client code based on OpenSSL, not EXPORT_RSA issues associated with servers or other TLS implementations.” [1]

CVE-2015-1637: “Schannel (aka Secure Channel) in Microsoft Windows Server 2003 SP2, Windows Vista SP2, Windows Server 2008 SP2 and R2 SP1, Windows 7 SP1, Windows 8, Windows 8.1, Windows Server 2012 Gold and R2, and Windows RT Gold and 8.1 does not properly restrict TLS state transitions, which makes it easier for remote attackers to conduct cipher-downgrade attacks to EXPORT_RSA ciphers via crafted TLS traffic, related to the "FREAK" issue, a different vulnerability than CVE-2015-0204 and CVE-2015-1067.” [2]

CVE-2015-1067: “Secure Transport in Apple iOS before 8.2, Apple OS X through 10.10.2, and Apple TV before 7.1 does not properly restrict TLS state transitions, which makes it easier for remote attackers to conduct cipher-downgrade attacks to EXPORT_RSA ciphers via crafted TLS traffic, related to the "FREAK" issue, a different vulnerability than CVE-2015-0204 and CVE-2015-1637.” [3]

[1] The MITRE Corporation, CVE--2015-0204, https://cve.mitre.org/cgi-bin/cvename.cgi?name=cve-2015-0204
[2] The MITRE Corporation, CVE--2015-1637, https://cve.mitre.org/cgi-bin/cvename.cgi?name=CVE-2015-1637.
[3] The MITRE Corporation, CVE--2015-1067, https://cve.mitre.org/cgi-bin/cvename.cgi?name=CVE-2015-1067.

BF: KMN Example (FREAK)

FREAK description using VRF taxonomy: An inner KMN leads to an inner ENC, which leads to an outer ENC.

Inner KMN:

- **Cause:** Improper Offer of Weak Protocol (Export RSA – offered from MITM-tricked server and accepted by client)
- **Attributes:**
  - Cryptographic Data: Keying Material (pair of private and public keys)
  - Data State: Transferred (over network)
  - Algorithm: Export RSA (512-bits key generation based on prime numbers, such that private key can be obtained from public key through factorization)
  - Operation: Generate
- **Consequence:** IEX Keying Material (private key)

Inner ENC:

- **Causes:** KMN Fault leads to Exposed Private Key
- **Attributes:**
  - Sensitive Data: Cryptographic (Pre-Master Secret)
  - Data State: Transferred (over network)
  - Algorithm: Asymmetric (RSA) (that allows decryption of Pre-Master Secret using exposed private key and computation of Master Secret)
  - Security Service: Confidentiality
- **Consequence:** IEX of Sensitive Data (Master Secret)

BF: KMN Example (FREAK)

Inner KMN and inner ENC only set up the secret key. Outer ENC is the actual general data transfer.

Outer ENC:

Causes: KMN Fault leads to Exposed Secret Key (Master Secret)
Attributes:
- Sensitive Data: Credentials (passwords, credit cards)
- Data State: Transferred (over network)
- Algorithm: Symmetric (key is known)
- Security Service: Confidentiality

Consequence: IEX of Sensitive Data (credentials)

BF: KMN Example (FREAK)

Interestingly in this example the consequence from the first bug (inner KMN) causes the second bug (inner ENC), whose consequences cause the third bug (outer ENC).

<u>Inner KMN</u> is:

- A server bug – sending a weak key (that the client did not ask for) intended for KMN use by client (encrypting Pre-Master Secret).
- And also a client bug – as the client accepted the offer of using the insecure method, and therefore the server proceeded. The client could have refused that offer.

<u>Inner ENC</u> is:

- A client bug – using that weak key to encrypt the Pre-Master Secret, and then transmitting that weakly encrypted Pre-Master Secret over a network that is not secure.

BF: KMN Example (FREAK) – Source Code

Client

`#ifndef OPENSSL_NO_RSA`	
`if (alg_k & SSL_kRSA) {`	`if (alg_k & SSL_kRSA) {`
	`  if (!SSL_C_IS_EXPORT(s->s3->tmp.new_cipher)) {`
	`    al=SSL_AD_UNEXPECTED_MESSAGE;`
	`SSLerr(SSL_F_SSL3_GET_SERVER_CERTIFICATE,SSL_R_UNEXPECTED_MESSAGE);`
	`    goto f_err;`
	`  }`
`if ((rsa=RSA_new()) == NULL) {`	`if ((rsa=RSA_new()) == NULL) {`
`SSLerr(SSL_F_SSL3_GET_KEY_EXCHANGE,ERR_R_MALLOC_FAILURE);`	`SSLerr(SSL_F_SSL3_GET_KEY_EXCHANGE,ERR_R_MALLOC_FAILURE);`

Server

`case SSL3_ST_SW_KEY_EXCH_B:`	`case SSL3_ST_SW_KEY_EXCH_B:`
`alg_k = s->s3->tmp.new_cipher->algorithm_mkey;`	`alg_k = s->s3->tmp.new_cipher->algorithm_mkey;`
`if ((s->options & SSL_OP_EPHEMERAL_RSA)`	
`#ifndef OPENSSL_NO_KRB5`	
`   && !(alg_k & SSL_kKRB5)`	
`#endif )`	
`    s->s3->tmp.use_rsa_tmp=1;`	
`else`	
`    s->s3->tmp.use_rsa_tmp=0;`	`s->s3->tmp.use_rsa_tmp=0;`
`if (s->s3->tmp.use_rsa_tmp`	`if (`

If client ciphersuit is non-export then returned by server RSA keys should be also non-export.

Therefore, handshake that offers export RSA key (512 bits, which is weak) should be abandoned by client.

The buggy code includes a handshake that enables accepting a 512-bit RSA key.

The fix is adding code that checks whether client ciphersuit is non-export and for abandoning the handshake if this is the case.

Access Control Cluster (ACC):

- Identity Proofing (IDP)
- Authentication Bugs (ATN)
- Authorization Bugs (ATZ)

<<still under development>>

Memory Cluster(ACC)

- Memory Allocation Bugs (MAL)
- Memory Use Bugs (MUS)

  Buffer Overflow (BOF) → refined

<<still under development>>

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.

  This definition is 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.

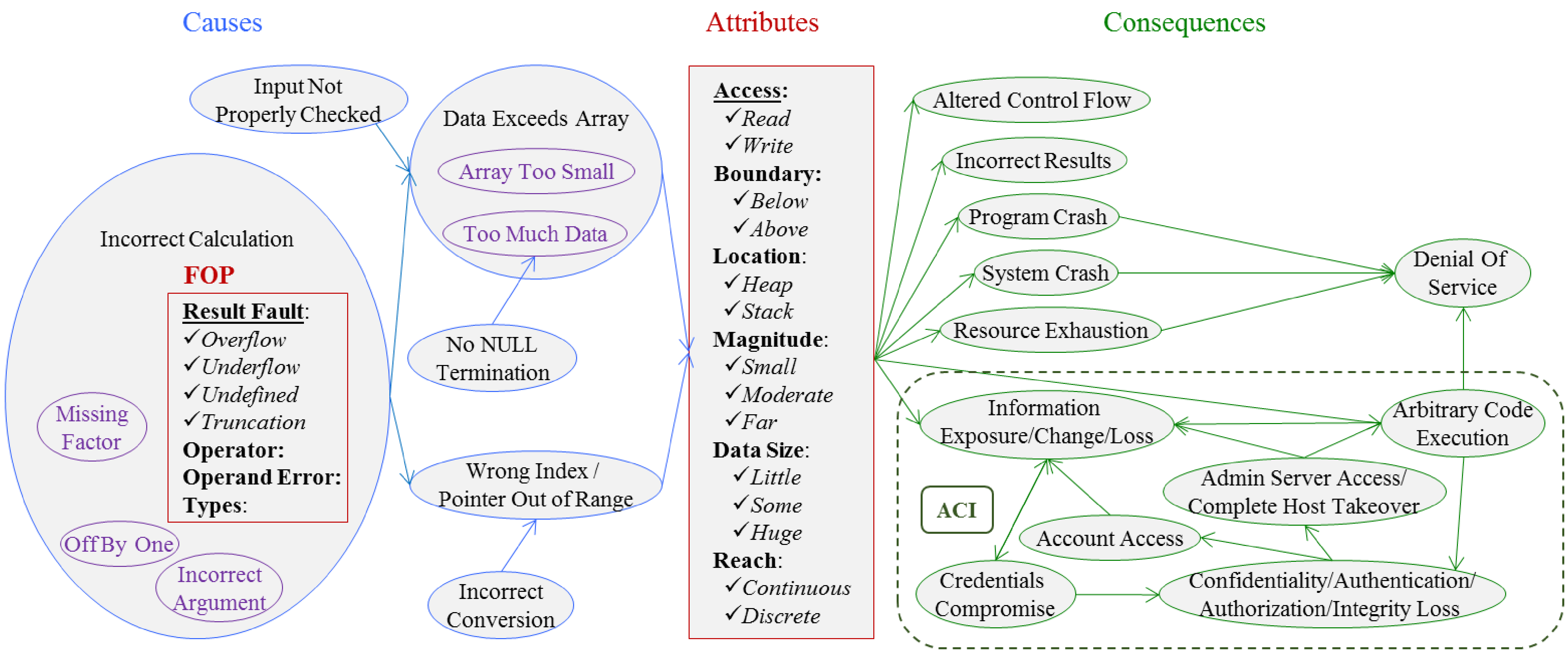
Related CWEs, SFP and ST

CWEs are 119, 120, 121, 122, 123, 124, 125, 126, 127, 786, 787, 788, 805, 806, 823.

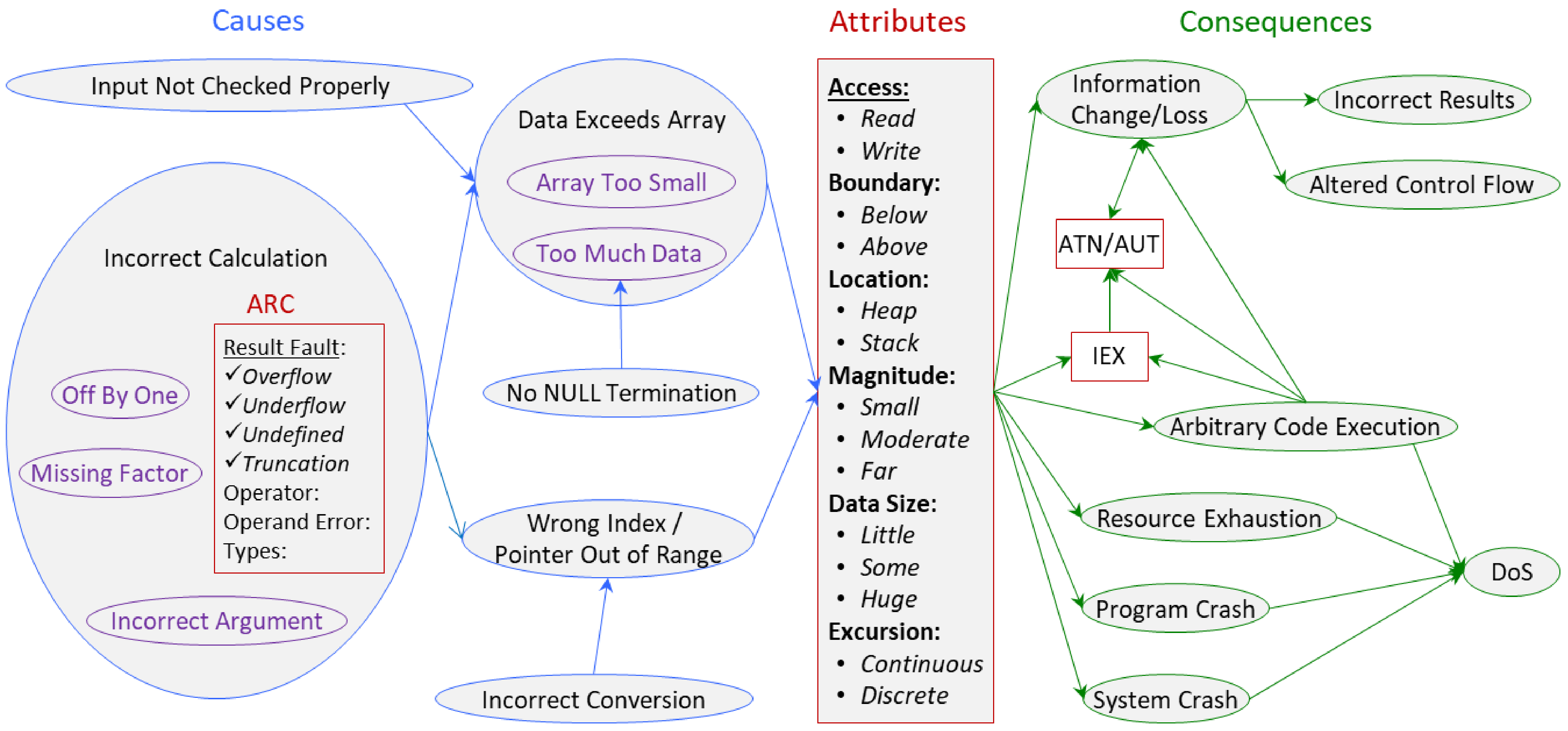
SFP cluster is SFP8 Faulty Buffer Access under Primary Cluster: Memory Access.

ST is the Buffer Overflow Semantic Template.

<<OLD>>BOF: Causes, Attributes, and Consequences



BOF: Causes, Attributes, and Consequences



Injection (INJ)

→ refined

BF: Injection (INJ)

- Our Definition:
  Due to input with language-specific special elements, the software assembles a command string that is parsed into an invalid construct.

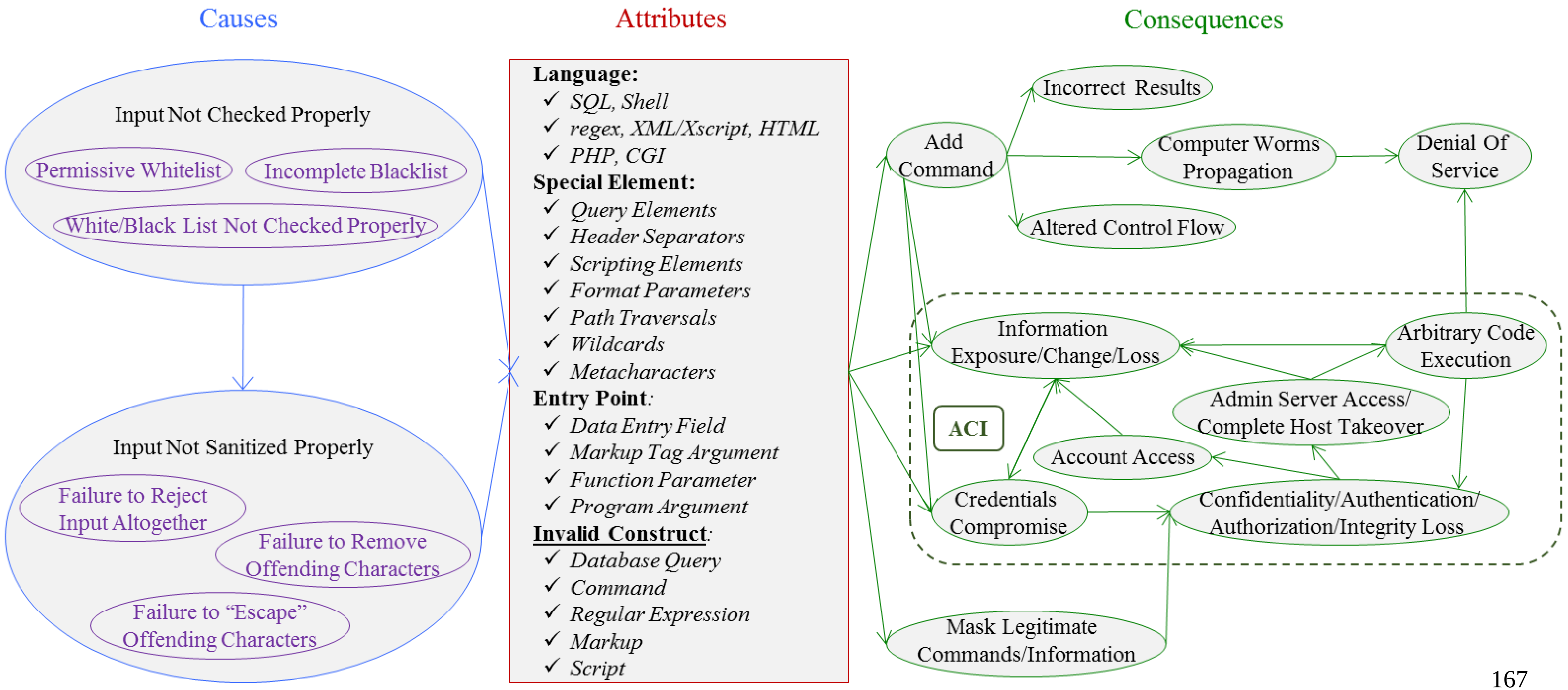
  In other words, the command string is interpreted to have unintended commands, elements or other structures.

  INJ is related to: ATN, ...

Related CWEs, SFPs and ST:

- ✓ CWEs related to INJ are 74, 75, 77, 78, 80, 85, 87, 88, 89, 90, 93,94, 243, 564, 619, 643, 652.
- ✓ Related SFPs are SFP24 and SFP27 under Primary Cluster: Tainted Input, and SFP17 under Primary Cluster: Path Resolution.
- ✓ The corresponding ST is the Injection Semantic Template.

<<OLD>>INJ: Causes, Attributes, and Consequences



Causes
Input Not Checked Properly
Permissive Whitelist
Incomplete Blacklist
White/Black List Not Checked Properly
Input Not Sanitized Properly
Failure to Reject Input Altogether
Failure to Remove Offending Characters
Failure to "Escape" Offending Characters
Attributes
Consequences
Incorrect Results
Add Command
Computer Worms Propagation
Denial Of Service
Altered Control Flow
Information Exposure/Change/Loss
Arbitrary Code Execution
Admin Server Access/ Complete Host Takeover
ACI
Account Access
Credentials Compromise
Confidentiality/Authentication/ Authorization/Integrity Loss
Mask Legitimate Commands/Information

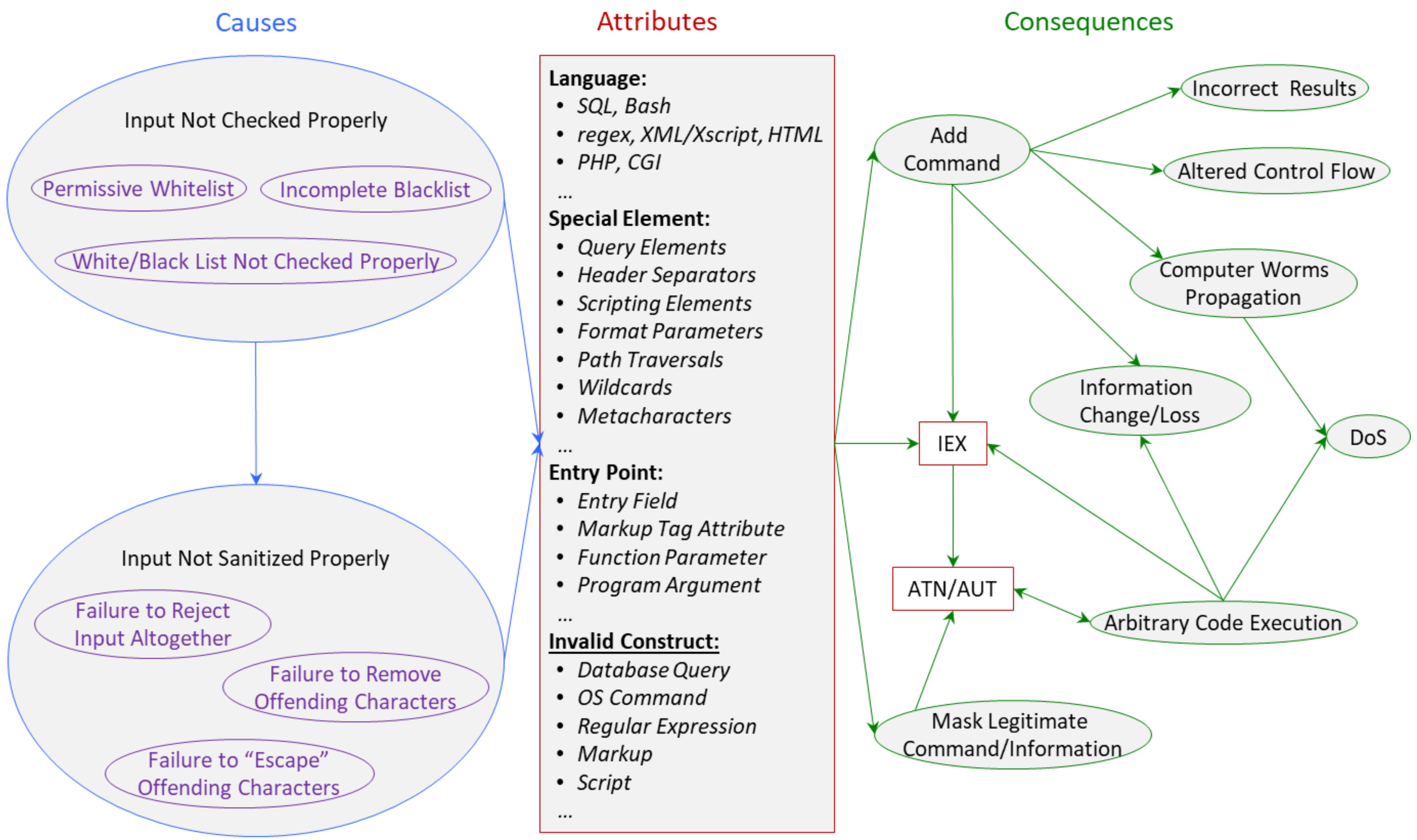
Language:
- ✓ *SQL, Shell*
- ✓ *regex, XML/Xscript, HTML*
- ✓ *PHP, CGI*

Special Element:
- ✓ *Query Elements*
- ✓ *Header Separators*
- ✓ *Scripting Elements*
- ✓ *Format Parameters*
- ✓ *Path Traversals*
- ✓ *Wildcards*
- ✓ *Metacharacters*

Entry Point:
- ✓ *Data Entry Field*
- ✓ *Markup Tag Argument*
- ✓ *Function Parameter*
- ✓ *Program Argument*

Invalid Construct:
- ✓ *Database Query*
- ✓ *Command*
- ✓ *Regular Expression*
- ✓ *Markup*
- ✓ *Script*

INJ: Causes, Attributes, and Consequences



Causes
Input Not Checked Properly
Permissive Whitelist
Incomplete Blacklist
White/Black List Not Checked Properly
Input Not Sanitized Properly
Failure to Reject Input Altogether
Failure to Remove Offending Characters
Failure to “Escape” Offending Characters
Attributes
Consequences
Incorrect Results
Add Command
Altered Control Flow
Computer Worms Propagation
Information Change/Loss
IEX
DoS
ATN/AUT
Arbitrary Code Execution
Mask Legitimate Command/Information

Language:
- *SQL, Bash*
- *regex, XML/Xscript, HTML*
- *PHP, CGI*

…

Special Element:
- *Query Elements*
- *Header Separators*
- *Scripting Elements*
- *Format Parameters*
- *Path Traversals*
- *Wildcards*
- *Metacharacters*

…

Entry Point:
- *Entry Field*
- *Markup Tag Attribute*
- *Function Parameter*
- *Program Argument*

…

Invalid Construct:
- *Database Query*
- *OS Command*
- *Regular Expression*
- *Markup*
- *Script*

…

Control of Interaction Frequency (CIF)

→ refined

BF: Control of Interaction Frequency (CIF)

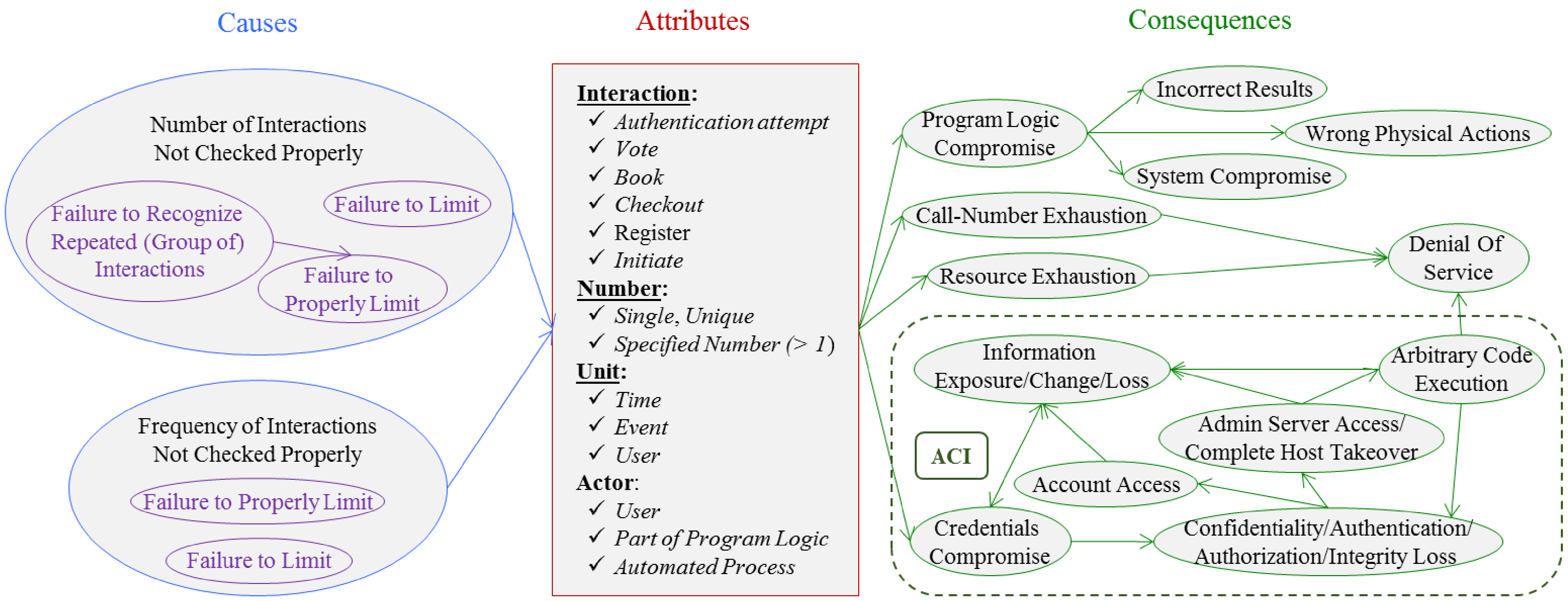
- Our Definition:
  The software does not properly limit the number of repeating interactions per specified unit.

  E.g. failed logins per day, one vote per voter per election (more for certain races!), maximum number of books checked out at once, etc. Interactions in software could be per event or per user.

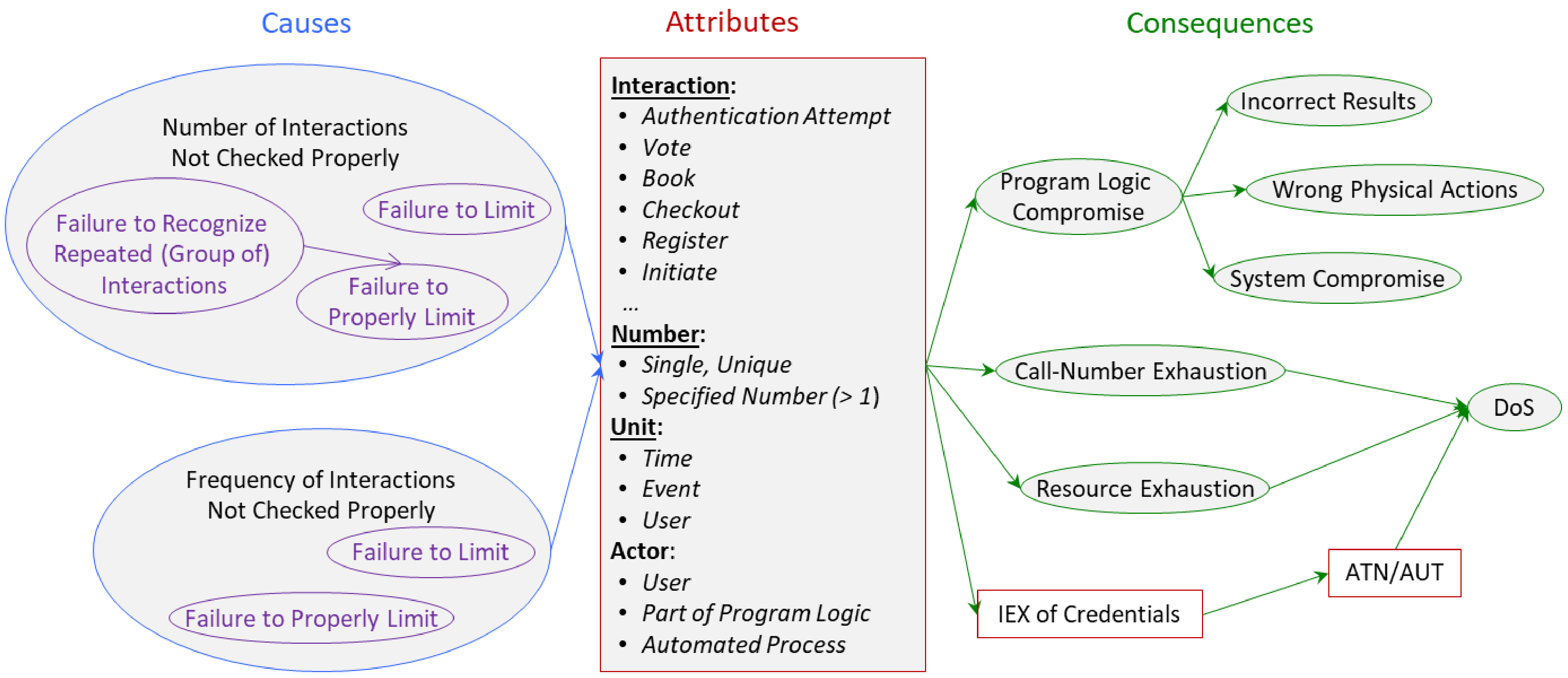
  This class shows that we must acknowledge outside or local "policies".

- Related CWEs, SFPs and ST:
  - ✓ CWEs related to CIF are 799, 307, 837.
  - ✓ The related SFP cluster is SFP34 Unrestricted Authentication under the Primary Cluster: Authentication.

<<OLD>>CIF: Causes, Attributes, and Consequences



CIF: Causes, Attributes, and Consequences



Interaction:
- *Authentication Attempt*
- *Vote*
- *Book*
- *Checkout*
- *Register*
- *Initiate*

…

Number:
- *Single, Unique*
- *Specified Number (> 1)*

Unit:
- *Time*
- *Event*
- *User*

Actor:
- *User*
- *Part of Program Logic*
- *Automated Process*

4. 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*.”

Questions

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