The Bugs Framework (BF)

Software Developers' and Testers' "Best Friend."

Irena Bojanova, Paul E. Black
National Institute of Standards and Technology (NIST)

Yaacov Yesha
National Institute of Standards and Technology (NIST)
University of Maryland Baltimore County (UMBC)

Yan Wu
Bowling Green State University (BGSU)

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 that compromise security
- What is meant by software having no known weaknesses?
- How to evaluate tools and services for finding weaknesses?
  → Need of classification of software weakness types

Bugs Terminology

- Software Weakness (Bug)

  "A piece of code that may lead to a vulnerability." [1]
- Security Vulnerability

  "A property of system requirements, design, implementation, or operation that could be accidentally triggered or intentionally exploited and result in a security failure." [1]
- Software Attack

  "The use of an exploit(s) by an adversary to take advantage of a weakness(s) with the intent of achieving a negative technical impact(s)." [2]

[1] Black, P., Kass, M., Koo. M., Fong, E. Source Code Security Analysis Tool Functional Specification Version 1.1, NIST Special Publication 500-268 v1.1.

[2] The MITRE Corporation. Common Attack Pattern Enumeration and Classification (CAPEC), Glosary, Attack.

Bugs Terminology

- Security Failure

  "Any event that is a violation of a particular system's explicit or implicit security policy." [1]

  - ✓ "the source of any failure is a latent vulnerability." [1]
  - ✓ "if there is a failure, there must have been a vulnerability." [1]

- Source Code

  "A series of statements written in a human-readable computer programming language." [1]

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

References

[1] Black, P., Kass, M., Koo. M., Fong, E. Source Code Security Analysis Tool Functional Specification Version 1.1, NIST Special Publication 500-268 v1.1.

Outline

1. Enlightenment
   - Repositories of Bugs, Vulnerabilities, and Attacks
   - Problems with Current Bug Descriptions
   - Need for Structured, Precise, Orthogonal Approach
2. The Bugs Framework (BF)
   - BF Taxonomy
   - Buffer Overflow (BOF)
   - Cryptography Classes (ENC, VRF, KMN)
3. Benefits of Using BF
   - Superior Unified Approach
   - Accurate, Precise, Clear Taxonomy
4. More BF Classes
   - Injection (INJ)
   - Faulty Result (FRS)
   - Control of Interaction Frequency Bugs (CIF)
   - Randomness Classes (RND, PRN)
5. Future Work

1. Enlightenment

Repositories of Bugs, Vulnerabilities, and Attacks

Repositories of Bugs, Vulnerabilities, and Attacks

BF is being created by factoring and restructuring of information contained in many existing repositories of bugs, vulnerabilities, and attacks and thus benefits from the community's experience with their use.

→ Let's take a look at them.

- ✓ 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)

Common Weakness Enumeration (CWE)

CWE is a “dictionary” of every *class* of bug or flaw in software.

More than 600 distinct classes, e.g.,

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

CWE is a community effort.

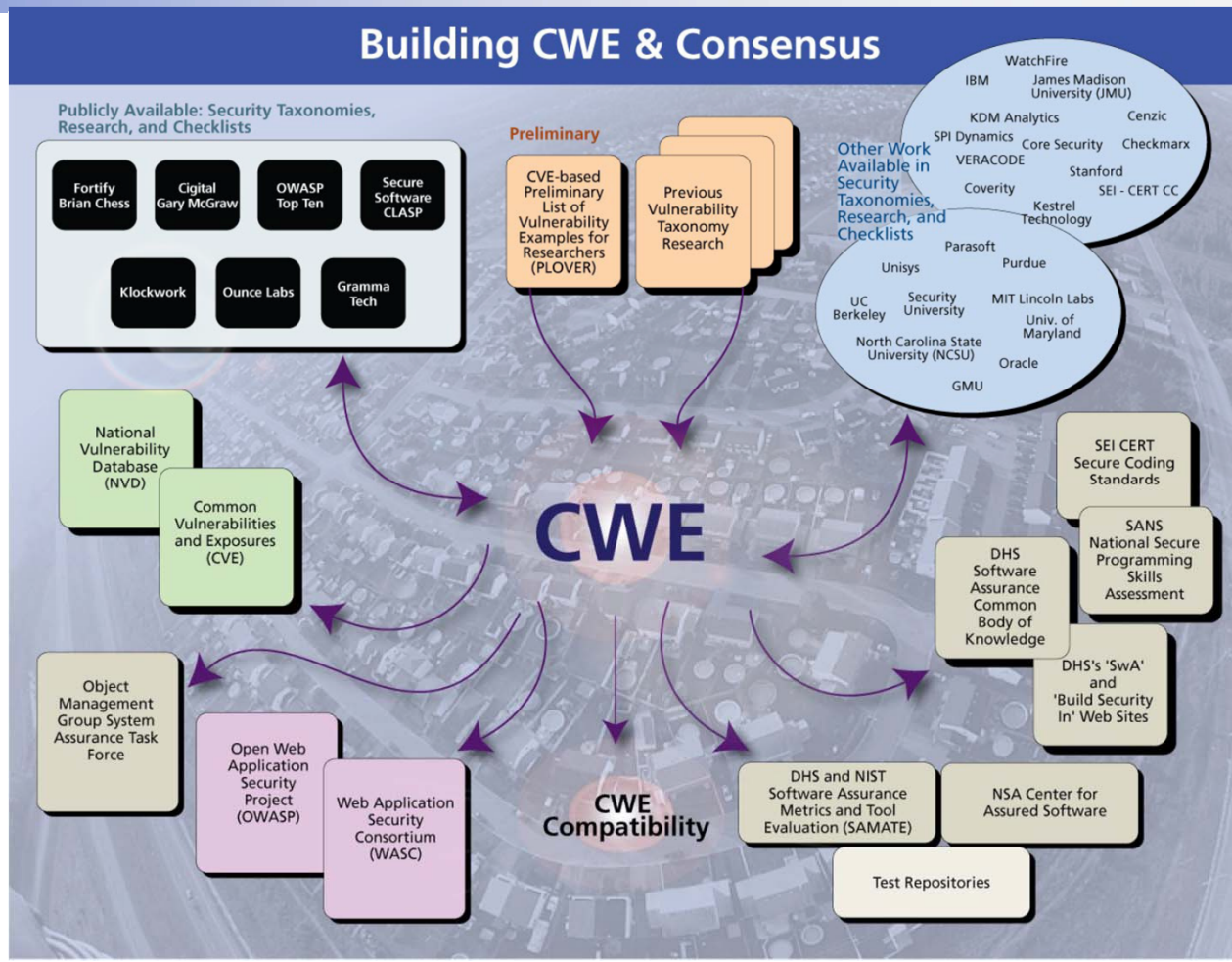


Fig. *CWE Efforts Context and Community* [http://cwe.mitre.org/about/images/lg_consensus.jpg]

Use of CWE

CWE – for use by those who:

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

CWE Compatibility and Effectiveness Program:

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

Designations for products or services:

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

Static analysis tools:

- also 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
- → So, the CWEs defined as weaknesses total 640.

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.

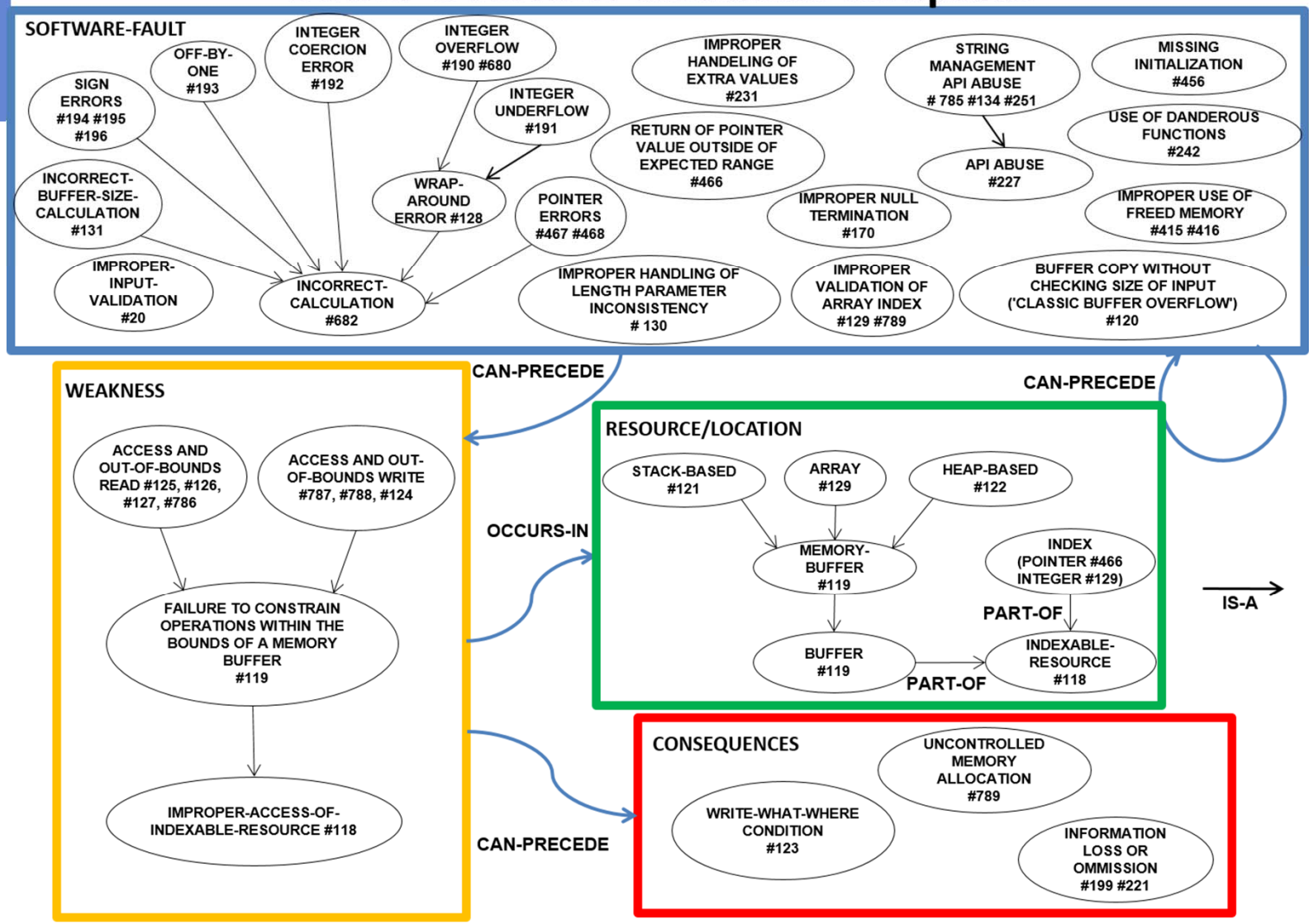
Extended Description: A buffer overflow condition exists when a program attempts to put more data in a buffer than it can hold, or when a program attempts to put data in a memory area outside of the boundaries of a buffer. The simplest type of error, and the most common cause of buffer overflows, is 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

Buffer Overflow Semantic Template



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:
  - ✓ Definitions are imprecise and inconsistent.
  - ✓ Entrees are "coarse grained" –
    bundle lots of stuff, like consequences and likely attacks.
  - ✓ The coverage is uneven –
    some combinations well represented and others not represented at all.
  - ✓ No mobile weaknesses, e.g., battery drain, physical sensors (GPS, gyro, microphone, hi-res camera), unencrypted wireless communication, etc.

CWE – Imprecise Definitions

- CWE-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!

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 – Gaps in Coverage

e.g. Buffer Overflow

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

versus

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

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

but

- *No reads from stack and heap.*

… while slight variants go on and on:

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

CWEs – Too Detailed

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

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

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.

Software Fault Patterns (SFP) – Improve on CWEs

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

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

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

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

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

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

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

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

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

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

Semantic Templates (ST) – Improve on CWEs, too

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

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

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

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

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

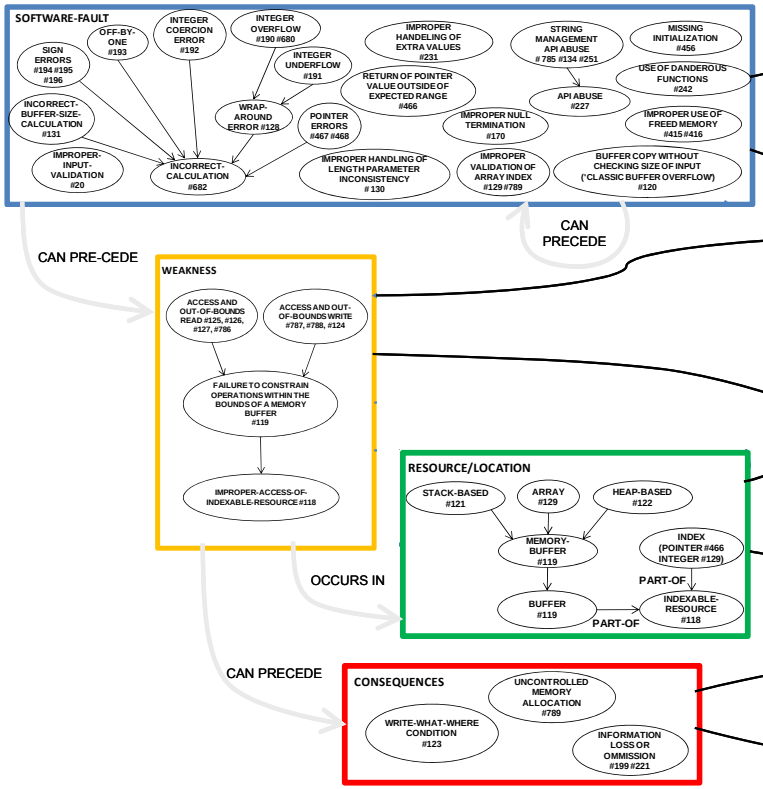
Summary: The program copies an input *buffer* to an output *buffer without verifying that the size of the input buffer is less than the size of the output buffer*, leading to a *buffer overflow*.

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

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

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

Semantic Templates (STs) – Improve on CWEs, too



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

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

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

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

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

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

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

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

But SFP & ST Also Have Problems

- Software Fault Patterns (SFP):
  - ✓ "Factor" weaknesses into parameters,
  - ✓ But:
    - Do not include upstream causes or consequences, and
    - Are based solely on CWEs.
- SFP is an excellent advance. However:

→ SFP does 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 guides to aid human comprehension.

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

Need for Structured, Precise, Orthogonal Approach

Need for Structured, Precise, Orthogonal Approach

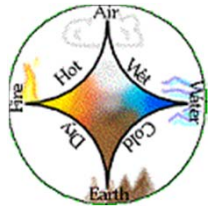
- Without accurate and precise classification and comprehension of all possible types of software bugs, the development of reliable software will remain extremely challenging.
- As a result the newly delivered and the legacy systems will continue having security holes despite all the patching to correct errant behavior.

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

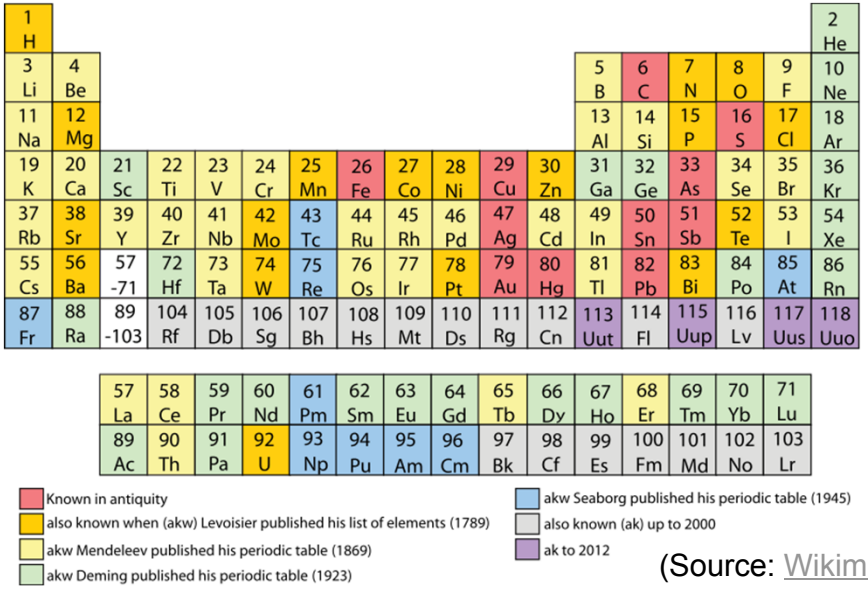
But, for analogies on what we are embarking on, let's look at some well-know organizational structures in science …

Periodic Table & Others to Describe Molecules

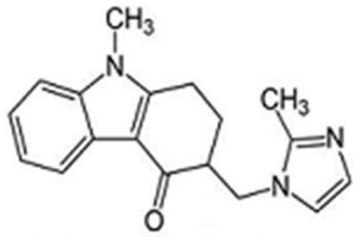
- Greeks used the terms element and atom.
  Aristotle: substances are a mix of Earth, Fire, Air, or Water.
- Alchemists cataloged substances, such as alcohol, sulfur, mercury, and salt.
  (note: Lavoisier had light and caloric on his 33 elements list!)
- Periodic table reflects atomic structure & forecasts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



(Source: Reich Chemistry)



(Source: Wikimedia Commons)



$C_{18}H_{19}N_3O$

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

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

Tree of Life

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

- Divides life into three domains:
  - ✓ Bacteria
  - ✓ Archaea
  - ✓ Eukaryotes.

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

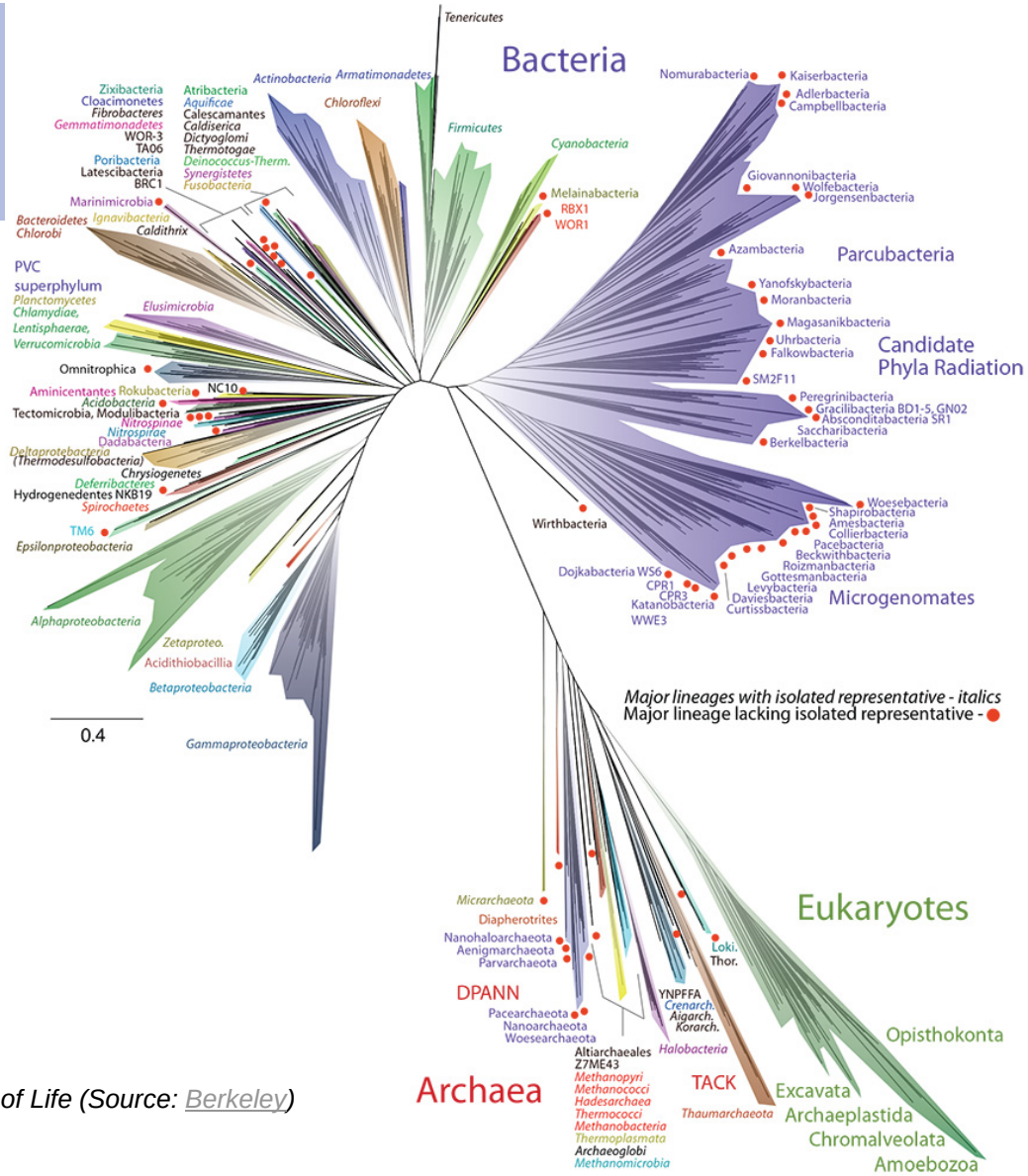
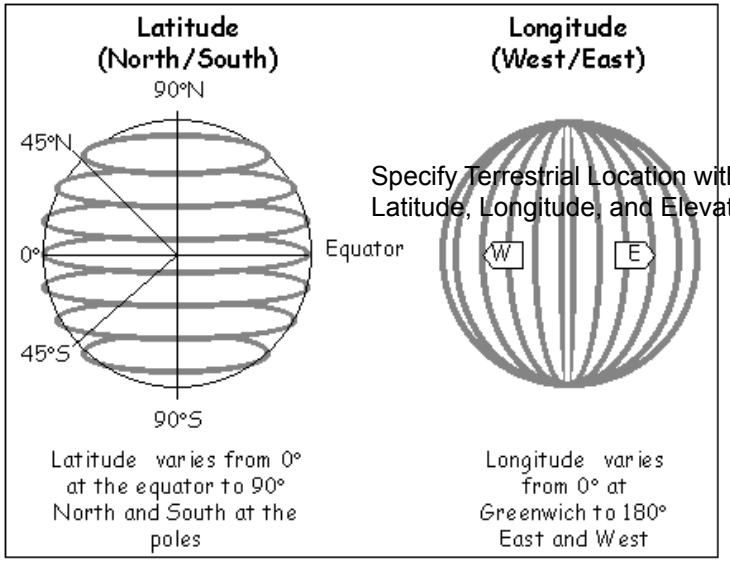
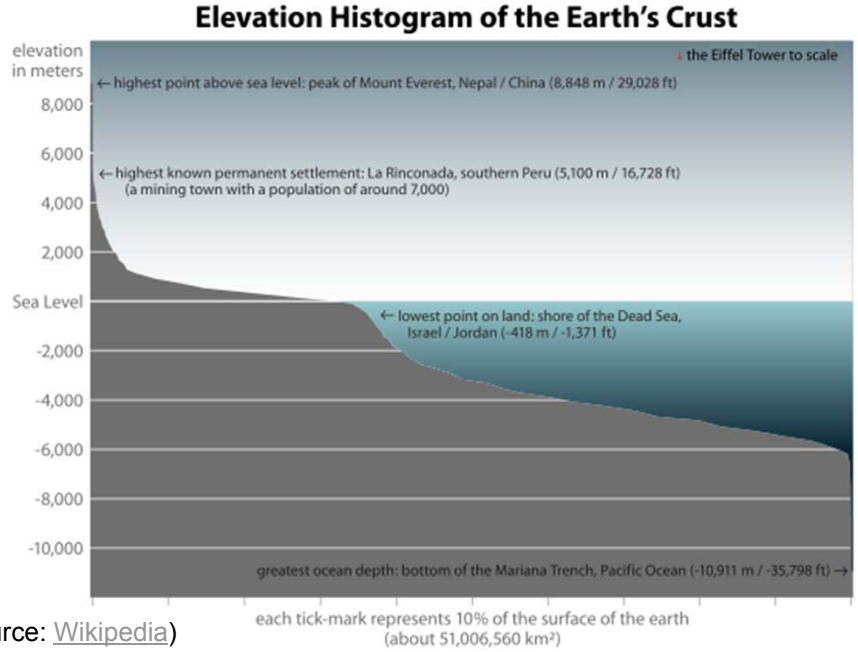


Fig. *The Tree of Life (Source: Berkeley)*

Geographic Coordinate System

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





Geographic Coordinate System (Source: Wikipedia)

Precise Medical Language

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

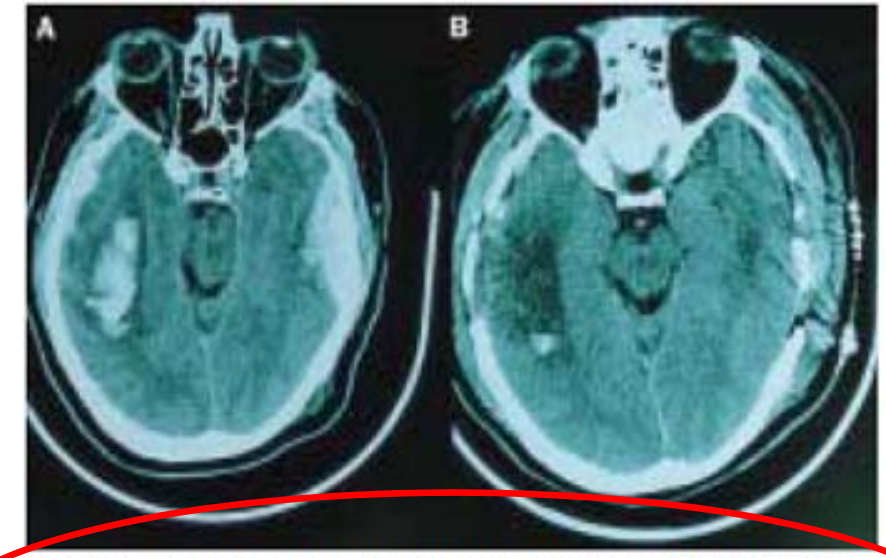


Figure 2: **Computed tomography of a comatose patient with a left temporal epidural haematoma, right parenchymal temporal lobe haematoma, and a right convexity subdural haematoma before and after craniotomy and evacuation of haematomas**

- 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)

2. The Bugs Framework (BF)

The Bugs Framework (BF)

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

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

BF Taxonomy

BF Taxonomy

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

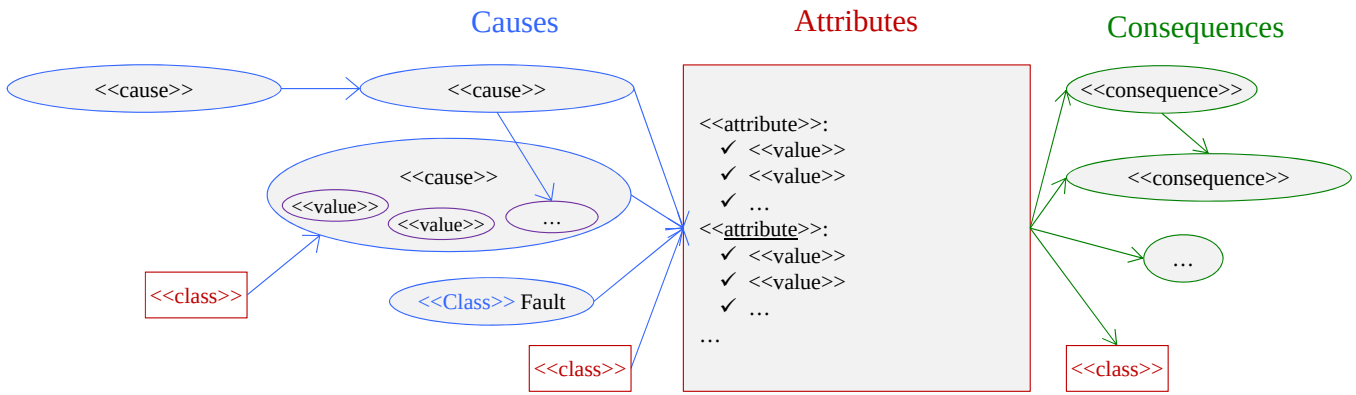
- Has an accurate and precise definition and
- Comprises:
  - ✓ Level (high or low) – identifies the fault as language-related or semantic.
  - ✓ Attributes – identify the software fault.
  - ✓ Causes – bring about the fault.
  - ✓ Consequences – to which the fault could lead.
  - ✓ Sites – locations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

- At least one attribute (underlined) identifies the software fault.
- Causes and consequences are directed graphs.
- Sites are identifiable mainly for low level classes
- BF uses precise definitions and terminology.

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

BF Class Graph

ClassName (ABR): <<consize definition>>.



Buffer Overflow (BOF)

Buffer Overflow

Buffer Overflow is the best class to begin with.

```
Char destination[5]; char *source = "LARGER";

strcpy(destination, source);
```

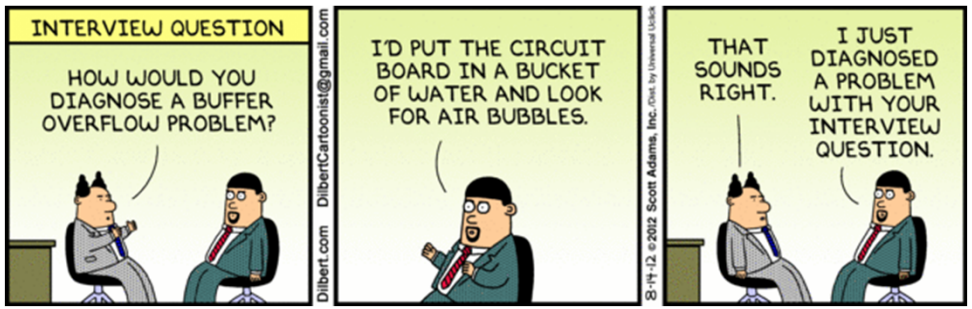
L	A	R	G	E	R	\0	

```
strncpy(destination, source, sizeof(destination));
```

L	A	R	G	E			

```
strlcpy(destination, source, sizeof(destination));
```

L	A	R	G	\0			



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 CWE-119, CWE-120, CWE-121, CWE-122, CWE-123, CWE-124, CWE-125, CWE-126, CWE-127, CWE-786, CWE-787, CWE-788.

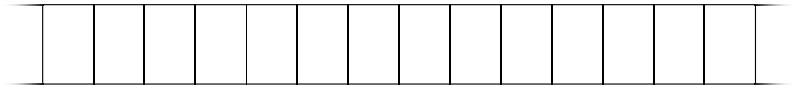
SFP cluster is SFP8 Faulty Buffer Access under Primary Cluster: Memory Access.

ST is the Buffer Overflow Semantic Template.

BOF Attributes

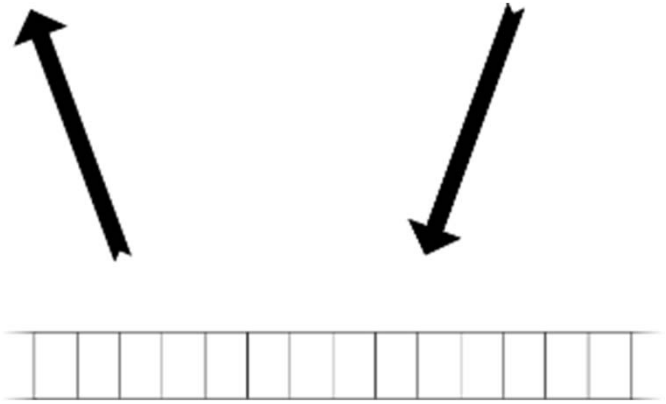
- Often referred to as a “buffer,” an array is a contiguously allocated set of objects, called elements.
  - ✓ Has a definite size – a definite number of elements are allocated to it.
  - ✓ Software should not use array name to access anything outside boundary of allocated elements.
  - ✓ Elements are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

An array could be pictured as follows:



BOF Attributes – Access

- Access: Read, Write.



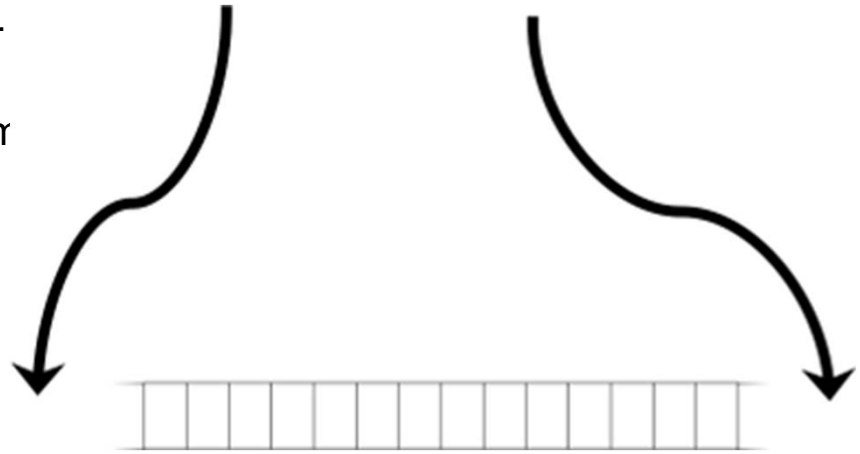
The underlined attribute shows what eventually goes wrong.

The rest of the attributes are simply descriptive.

BOF Attributes – Boundary

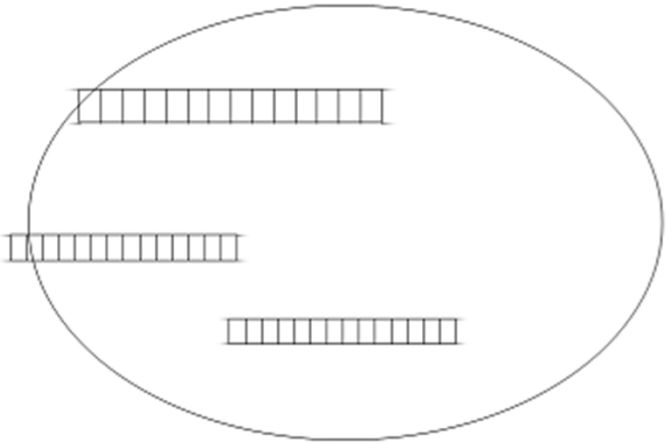
- Access: Read, Write.
- **Boundary** – indicates which end of the array is violated:
  - ✓ Below (before, under, or lower)
  - ✓ Above (after, over, or upper).

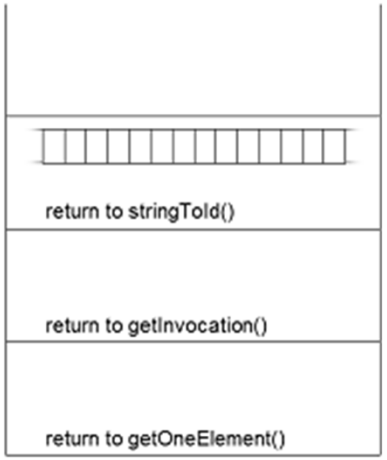
- Synonyms for boundary are side or bound.
- Before, under or lower may be used instead of below.
- After, over or upper may be used instead of above.
- Outside indicates boundary is unknown or it doesn't m



BOF: Attributes – Location

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

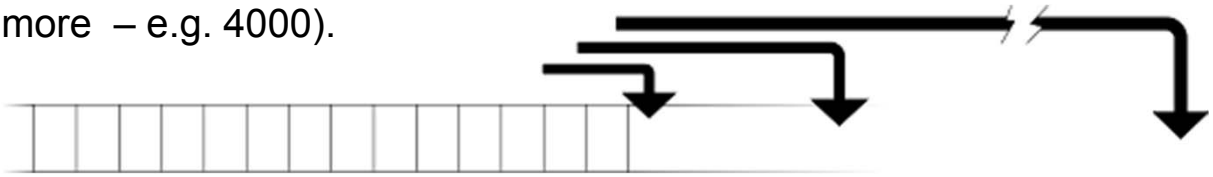




- It may matter since:
  - violations in the stack may affect program execution flow
  - while violations in the heap typically only affect data values.
- Other compilers and operating system may have other locations that are significant
  - e.g. BSS, Data, Code (text).

BOF Attributes – Magnitude

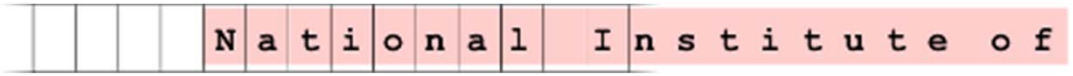
- Access: Read, Write.
- Boundary: Below, Above.
- Location: Heap, Stack, BSS, Data, Code.
- **Magnitude** – how far outside the boundary the violation extends:
  - ✓ Small (just barely outside – e.g. one to a few bytes)
  - ✓ Moderate (from 8 to dozens of bites)
  - ✓ Far (hundreds, thousands or more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



These distinctions in the magnitude attribute are important because some violation detection techniques or mitigation techniques, such as canaries or allocating a little extra space, are only useful if the magnitude is small.

BOF Attributes – Data Size

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



As in magnitude, these distinctions are important in some cases

– e.g. 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.

BOF Attributes – Excursion

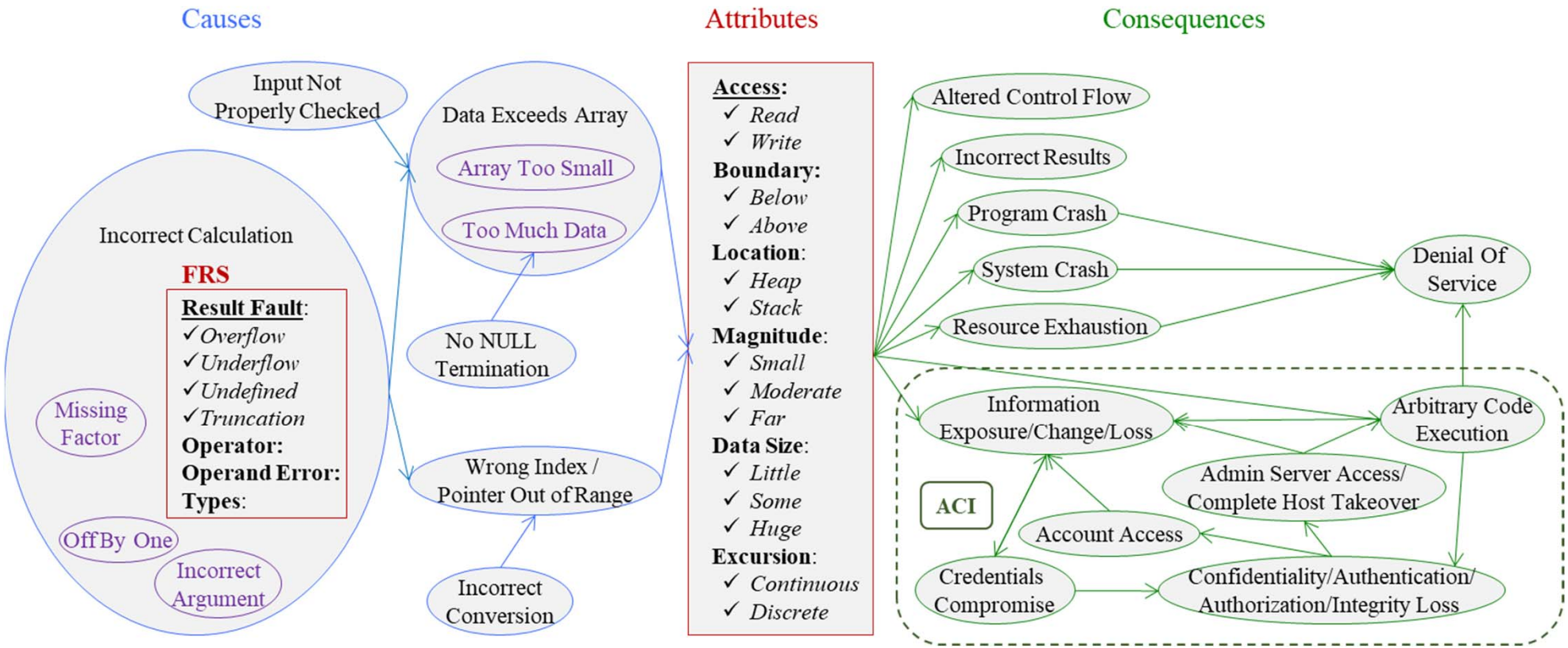
- Access: Read, Write.
- Boundary: Below, Above.
- Location: Heap, Stack, BSS, Data, Code.
- Magnitude: Small, Moderate, Far.
- Data Size: Little, Some, Huge.
- **Excursion** – one-by-one or arbitrary:
  - ✓ Continuous
  - ✓ Discrete.

This indicates whether the access violation was preceded by consecutive access of elements starting within the array (continuous) or just an access outside of the array (discrete). Typically string accesses or array copies handle a continuous set of array elements, while a vagrant array index only reads or writes one element.

→ All attributes can also be “either/any/don’t care/unknown”.

For instance, strict bounds checking is equally effective regardless of the location, magnitude, data size or excursion of the violation. Keeping return addresses in a separate stack helps prevent problems occurring from write accesses when the array location is the stack.

BOF: Causes, Attributes, and Consequences



BOF: Sites

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

Cryptography Classes in BF (ENC, VRF, KMN)

Cryptography Classes in BF

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

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.

Cryptographic Store or Transfer

We use cryptographic store or transfer to illustrate the BF Cryptography Bugs 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:

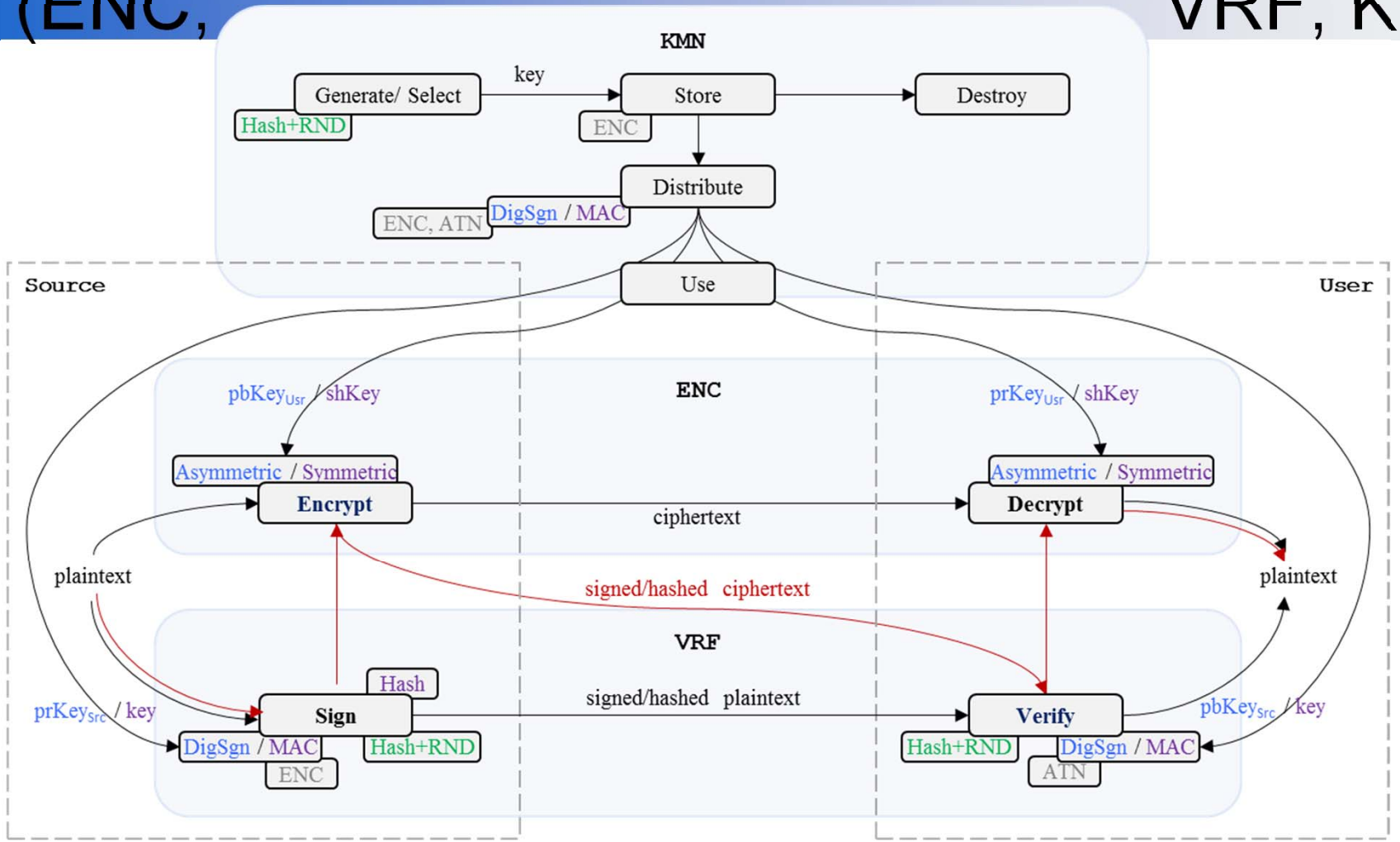
- ✓ Well known a
- ✓ What most people think of when “cryptography” is mentioned.

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.

Model of Cryptographic Store or Transfer Bugs (ENC, VRF, KMN)



Model: Key Management Bugs (KMN)

- 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.

Model: Key Management Bugs (KMN)

- 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.

Model: Encryption Bugs (ENC)

- 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.

Model: Verification Bugs (VRF)

- 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.

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:
    - o encrypts with $pbKey_{Usr}$
    - o signs with $prKey_{Src}$
  - ✓ User:
    - o decrypts with $prKey_{Usr}$
    - o verifies with $pbKey_{Src}$.

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).

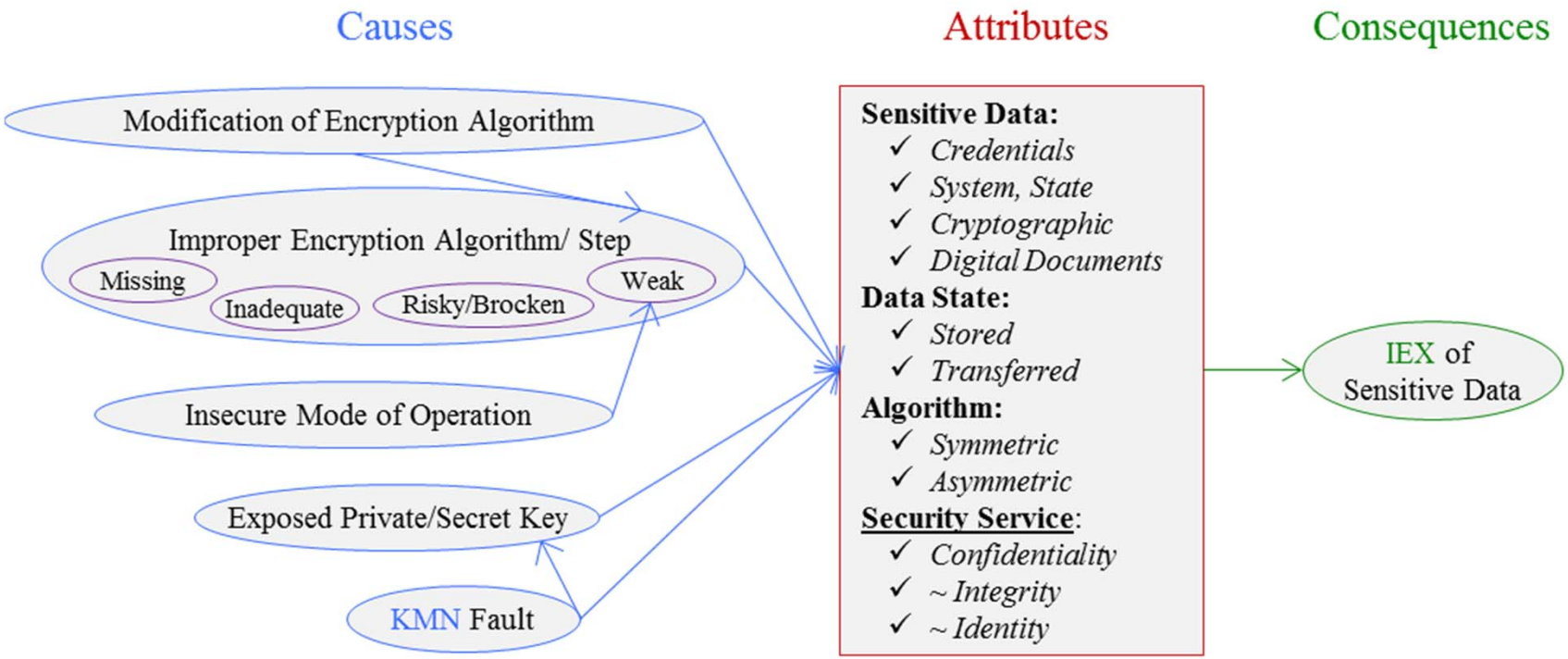
Note that "transform" is for confidentiality.

ENC is related to KMN, Randomization (RND), and Information Exposure (IEX).

Related CWEs, SFPs and ST:

- ✓ CWEs: CWE-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



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.

Verification Bugs (VRF)

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

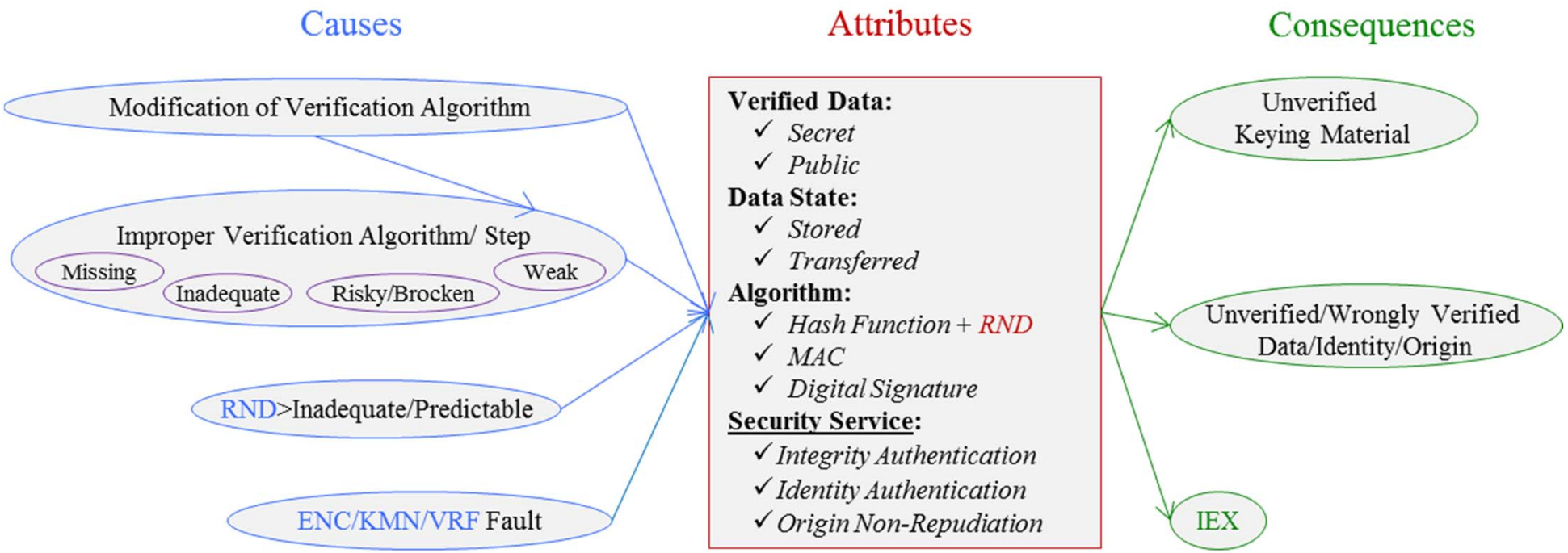
  Note that "check" is for identity authentication, "prove" is for origin (signer) non-repudiation, and "not altered" is for integrity authentication.

  VRF is related to KMN, RND, ENC, Authentication (ATN), IEX.

Related CWEs, SFPs and ST:

- ✓ CWEs: CWE-295, 296, 347.
- ✓ SFP cluster: SFP 17.2 Weak Cryptography under Primary Cluster: Cryptography.

VRF: Causes, Attributes, and Consequences



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.

Key Management Bugs (KMN)

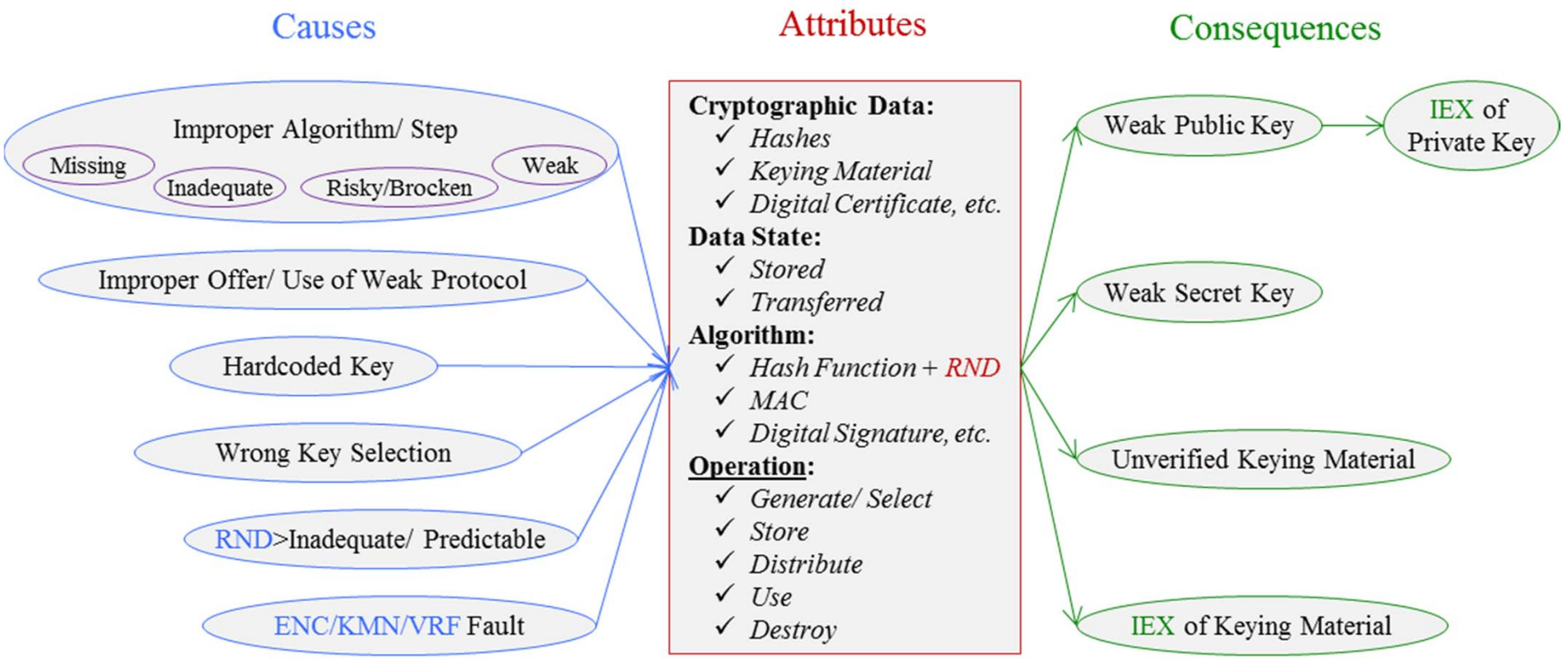
- Our Definition:
  The software does not properly generate, store, distribute, use, or destroy cryptographic keys and other keying material.

  KMN is related to ENC, RND, VRF, IEX.

Related CWEs, SFPs and ST:

- ✓ CWEs: CWE-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.
- **Operation** –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.

3. 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.

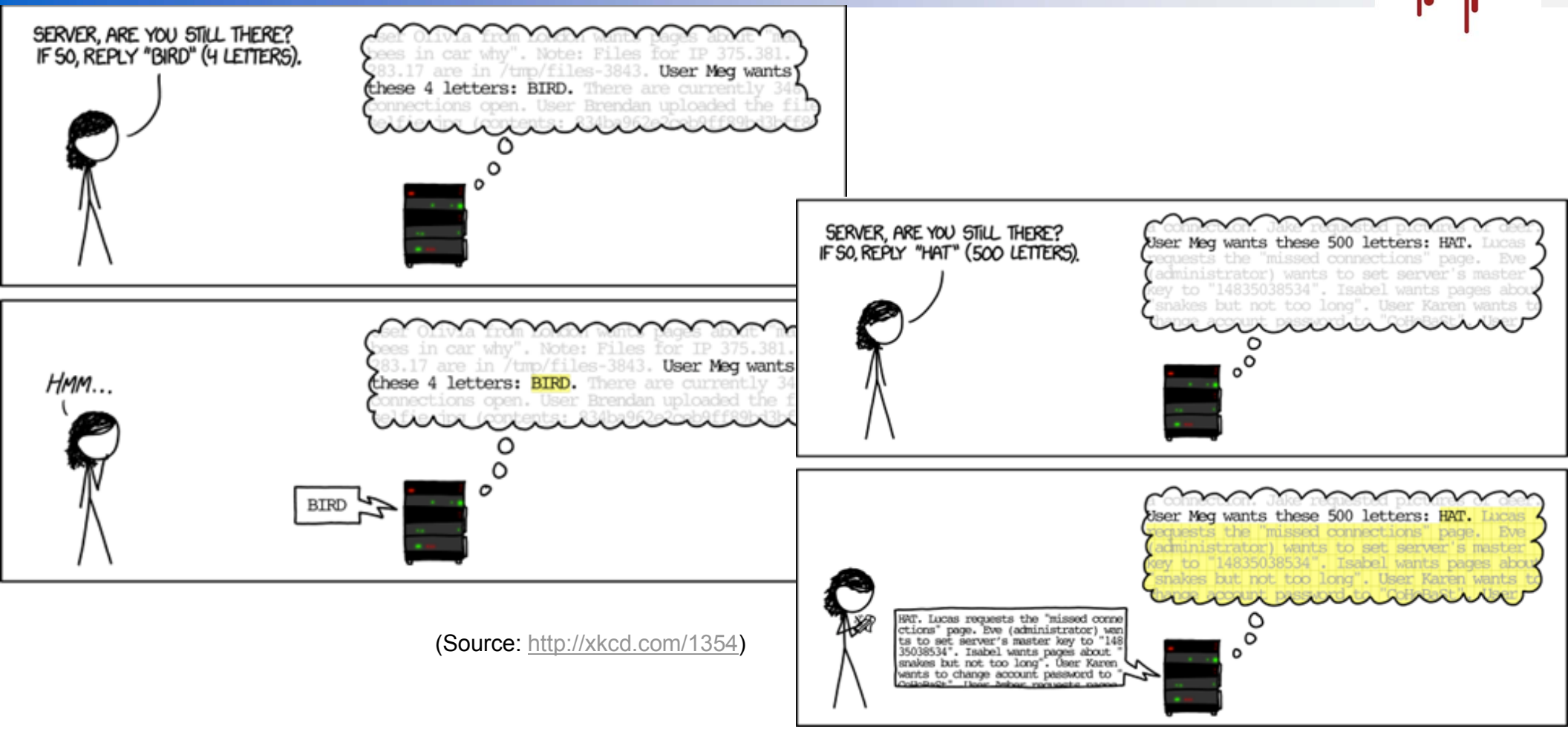
- Those responsible for designing, operating and maintaining computer complexes
  → can communicate with more exactness about threats, attacks, patches and exposures.

BOF Examples

BOF Example 1 – BF Explains Techniques

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

BOF Example 2 – CVE-2014-0160 (Heartble



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

BOF Example – CVE-2014-0160 (Heartbleed)

BOF taxonomy:

Cause: Input Not Checked Properly leads to Data Exceeds Array (specifically Too Much Data)

Attributes:

Access: Read

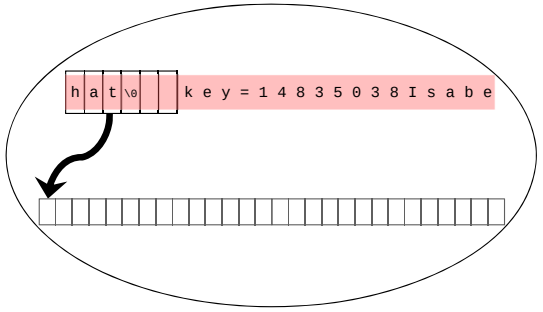
Boundary: Above

Location: Heap

Data Size: Huge

Excursion: Continuous

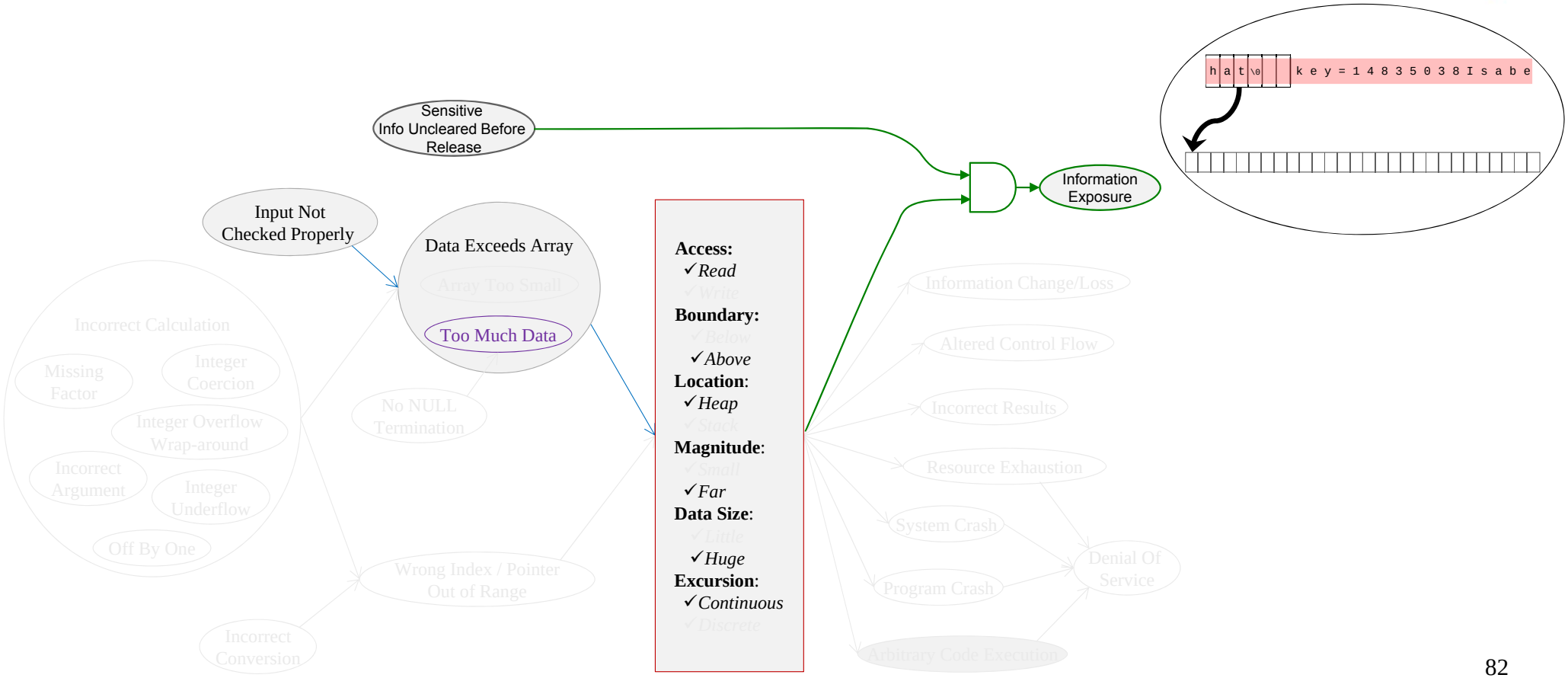
Consequence: IEX (if not had been cleared - *CWE-226*)



BF description: Input not checked properly leads to data exceeds array (specifically too much data), where huge data is read from the heap in a continuous excursion above the array boundary, which may be exploited for IEX (if not had been cleared)."

CVE-2014-0160 (Heartbleed): "*The (1) TLS and (2) DTLS implementations in OpenSSL 1.0.1 before 1.0.1g do not properly handle Heartbeat Extension packets, which allows remote attackers to obtain sensitive information from process memory via crafted packets that trigger a buffer over-read, as demonstrated by reading private keys, related to d1_both.c and t1_lib.c, aka the Heartbleed bug."*

BOF Example 2 – CVE-2014-0160 (Heartble



ENC, VRF, KMN Examples

ENC Example – CVE-2002-1946

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)

BF description: Use of weak encryption algorithm (one-to-one mapping) allows obtaining the shared symmetric key and decryption of stored (in registry) credentials (passwords), which is confidentiality failure and IEX of sensitive data (passwords).

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.

VRF: Example – CVE-2001-1585

VRF taxonomy:

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

BF description: *Missing verification step (challenge-response) in public key authentication allows private key for digital signature not to be verified by public key, which leads to identity authentication failure and may be exploited for IEX.*

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.

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.

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)

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)

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).

Inner KMN 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.

Inner ENC 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.

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.

KMN Example (FREAK) – Analysis

The following analysis is based on information in [1-7].

- The server offers a weak protocol (Export RSA) while the client requested strong protocol (RSA).
- Communication is encrypted by symmetric encryption. The key for that encryption (Master Secret) is created by both client and server from a Pre-Master Secret and nonces sent by client and server. The Pre-Master Secret is sent encrypted by RSA cryptosystem.
- The client requests RSA protocol, but man in the middle (MITM) intercepts and requests Export RSA that uses a 512 bit key. Factoring a 512 bit RSA key is feasible.
- Because of a bug, the client agrees to Export RSA.
- MITM factors the public 512 bit public RSA key, uses this factoring to recover the private RSA key, and then uses that private key to decrypt the Pre-Master Secret.
- Then it uses the Pre-Master Secret and the nonces to generate the Master Secret. The Master Secret enables MITM to decrypt the encrypted communication from that point on.

Note: For Export RSA, a weaker RSA key-pair (512-bit) is required than required on the SSL certificate. If it was RSA, the client would generate the Pre-Master Secret and encrypt it with server's public key (min 1024-bit) from its SSL certificate.

[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.

[4] R. Heaton, The SSL FREAK vulnerability explained, http://robertheaton.com/2015/04/06/the-ssl-freak-vulnerability.

[5] Censys, The FREAK Attack. https://censys.io/blog/freak

[6] StackExchange, Protecting phone from the FREAK bug, http://android.stackexchange.com/questions/101929/protecting-phone-from-the-freak-bug/101966.

[7] GitHub, openssl, Only allow ephemeral RSA keys in export ciphersuites, https://github.com/openssl/openssl/commit/ce325c60c74b0fa784f5872404b722e120e5cab0?diff=split.

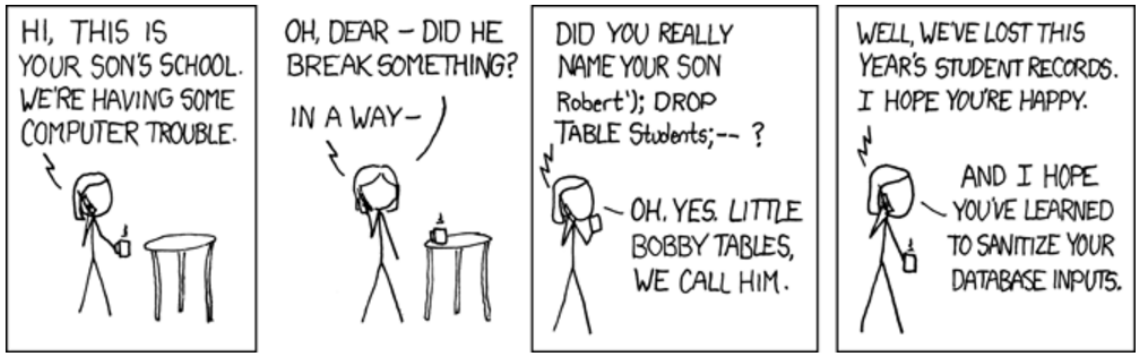
4. More BF Classes

Developed BF Classes

- Buffer Overflow (BOF)
- Injection (INJ), e.g.
  - ✓ SQL injection
  - ✓ OS injection.
- Control of Interaction Frequency Bugs (CIF),e.g.
  - ✓ Limit number of login attempts
  - ✓ Only one vote per voter.
- Encryption Bugs (ENC)
- Verification Bugs (VRF)
- Key Management Bugs (KMN)
- Faulty Result (FRS)
- Random Number Generation Bugs (RND)
- Pseudo-Random Number Generation Bugs (PRN).

Injection

- What is Injection? (in programming, not in medicine ☺)



Source: xkcd

Injection (INJ)

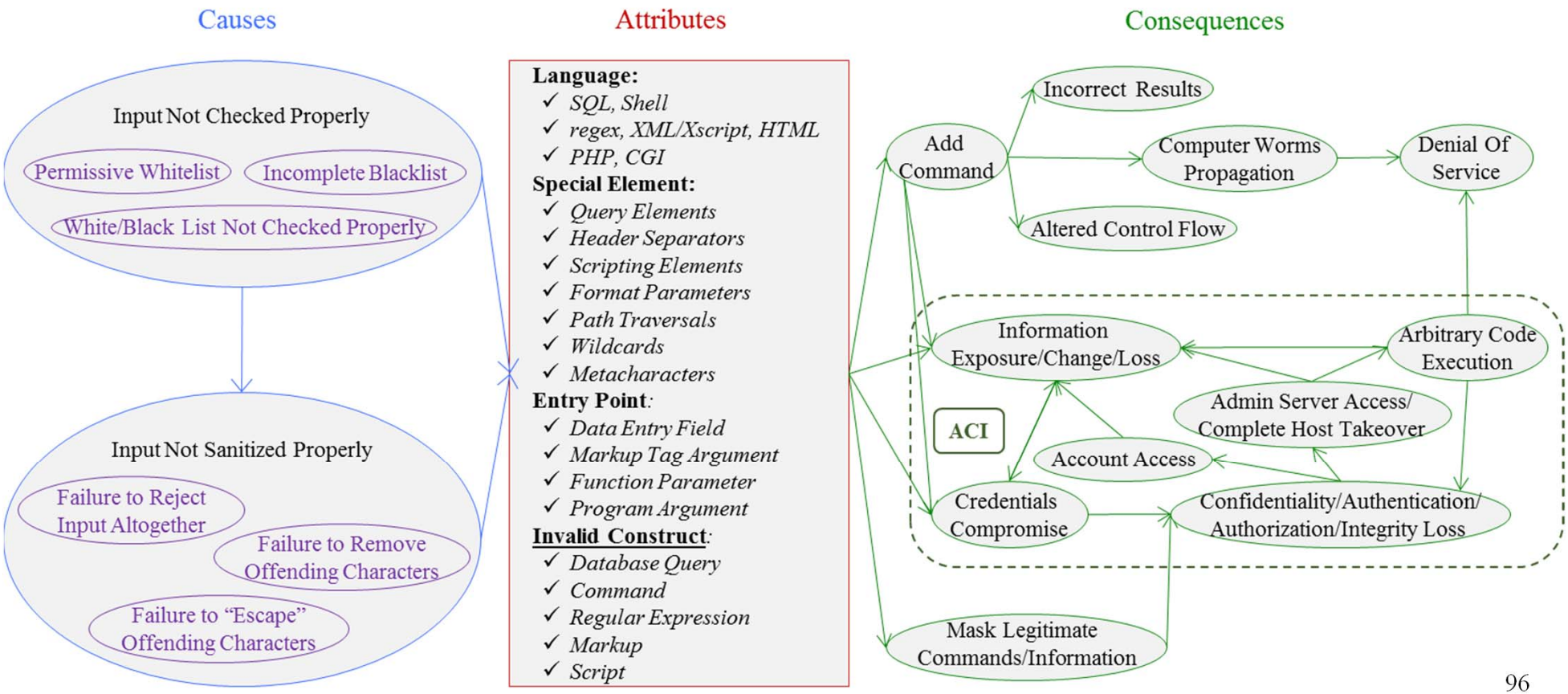
- Our Definition:
  Due to input with language-specific special elements, the software assembles a command string that is parsed into an invalid construct.

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

Related CWEs, SFPs and ST:

- ✓ CWEs related to INJ are CWE-74, CWE-75, CWE-77, CWE-78, CWE-80, CWE-85, CWE-87, CWE-88, CWE-89, CWE-90, CWE-93,CWE-94, CWE-243, CWE-564, CWE-619, CWE-643, CWE-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.

INJ: Causes, Attributes, and Consequences



INJ: Attributes

- **Language** – in which the command string is interpreted:
  SQL, Bash, regex, XML/Xpath, PHP, CGI, etc.
- **Special Element** – could be assembled with other elements to form malicious structures such as queries, scripts and commands:
  - ✓ Query Elements: strings delimiters ‘ or “ or words such as ‘and’ or ‘or’.
  - ✓ Header Separators: carriage return/line feed.
  - ✓ Scripting Elements: < or > or &.
  - ✓ Format Parameters: such as %c or %n.
  - ✓ Path Traversals Element: .. or \.
  - ✓ Metacharacters: back tick (`) or $ or &.
- **Entry Point** – where the input came from:
  Data Entry Field, Scripting Tag, Markup Tag, Function Call Parameter, Procedure Call Argument.
- **Invalid Construct** – what eventually is wrong:
  Database Query, Command, Regular Expression, Markup, Script, etc.
  (correspond to the note after the definition: "the command string is interpreted to have unintended commands, elements or other structures".)

Control of Interaction Frequency (CIF)

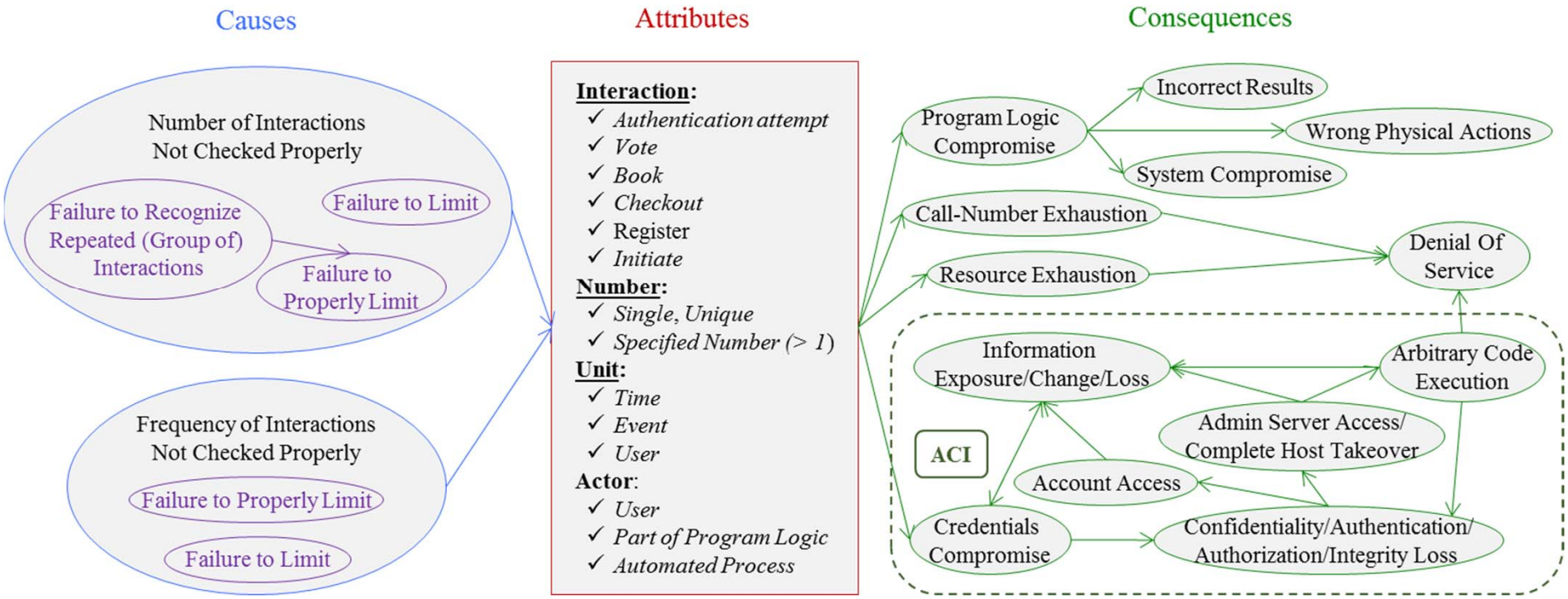
- Our Definition:
  The software does not properly limit the number of repeating interactions per specified unit.

  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 CWE-799, CWE-307, CWE-837.
  - ✓ The related SFP cluster is SFP34 Unrestricted Authentication under the Primary Cluster: Authentication.

CIF: Causes, Attributes, and Consequences



CIF: Attributes

- **Interaction –** to be controlled:
  - Authentication Attempt
  - Vote – related to election, census, survey, referendum and ballot
  - Book – tickets, hotel rooms or rental cars.
  - Checkout – of library books, hotel rooms or rental cars.
  - Register – computer games
  - Initiate – message exchange.
- **Number –** maximum occurrences allowed:
  - Single, Unique; Specified Number (> 1).
- **Unit –** per which the number of occurrences is controlled.
  - Time Interval – in seconds, in days, etc.
  - Event – election, authentication, on-line transaction to move funds, etc.
- **Actor –** who/what is performing the repeating interactions:
  - User – authenticated user, attacker.
  - Part of program logic – message exchange.
  - Automated process – virus, bot.

Faulty Results (FRS)

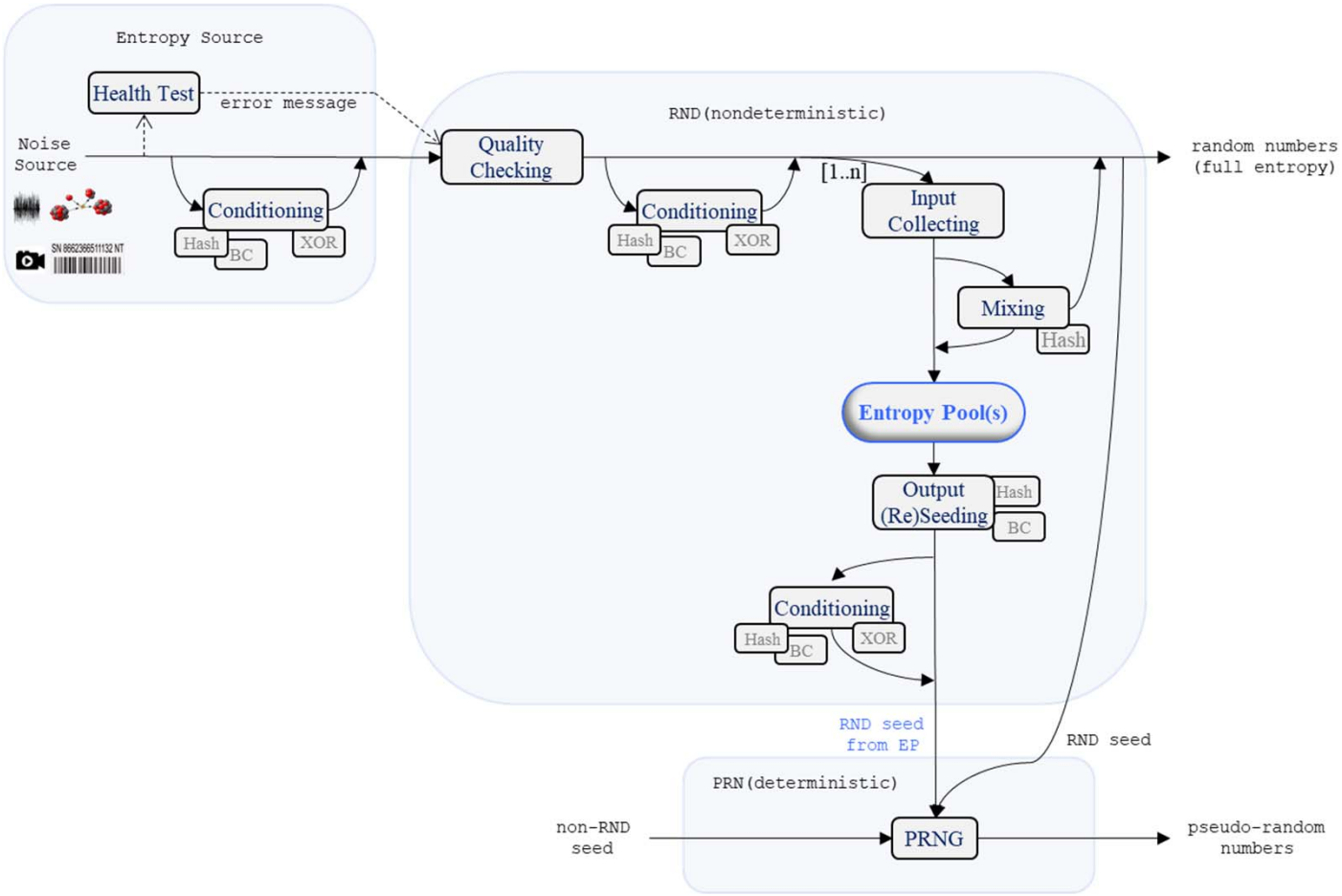
- Our Definition:
  The software produces a faulty result due to conversions between primitive types, range violations, or domain violation.
  - *The terms "range violation" and "domain violation" include overflow, underflow, wrap around, round off, divide by zero, and negative shift. Overflow is when the combined value of the two operands exceeds the range of the typed operator [1].*
  - *Operations on pointers (memory addresses) are **not** FRS [2]. Use of wrong typed operation, e.g. integer division instead of floating point division, is WOP, **not** FRS.*
  - *Floating point overflow and underflow, conversion between floating point types, and conversion between floating point and integer types **are** FRS. Numerical analysis errors, such as loss of precision, failure to converge, and conditioning problems, are **not** FRS.*

The model is that an operation causes (implicit) conversion of its operands' values, then the operation is performed. How does argument passing and the C cast fit this model? Argument passing can be seen as an implicit conversion then a null operation (or, equivalently, an identity operation is performed). The C cast can be seen as explicit conversion then null.

Related CWEs, SFPs and SEI/CERT Rules and Recommendations:

- CWEs related to FRS are CWE-128, CWE-136, CWE-189, CWE-190, CWE-191, CWE-192, CWE-194, CWE-195, CWE-196, CWE-197, CWE-369, CWE-681, CWE-682, CWE-704.
- Related SFP secondary cluster is Glitch in Computation (CWE-998).
- Related SEI/CERT Rules and Recommendations are Rule 04 (C), (aka CWE-738), INT30-C, INT31-C, INT32-C, INT33-C, INT34-C, INT35-C, INT36-C; Rule 5 (C) (aka CWE-739), FLP03-C, FLP32-C, FLP34-C, FLP36-C; Rule 07 (C) (aka CWE-741), STR34-C, STR37-C; Rec. 04 (C) INT01-C, INT02-C, INT04-C, INT05-C, INT07-C, INT08-C, INT10-C, INT12-C, INT13-C, INT14-C, INT15-C, INT16-C, INT18-C; Rec. 08 (C) (aka CWE-742), MEM07-C; Rule 03 (C++) (aka CWE-872), INT50-CPP; Rule 03 (Java) (aka CWE-848), NUM00-J, NUM01-J, NUM02-J, NUM03-J, NUM04-J, NUM08-J, NUM12-J, NUM13-J, NUM14-J.

Model of Randomness Bugs (RND, PRN)



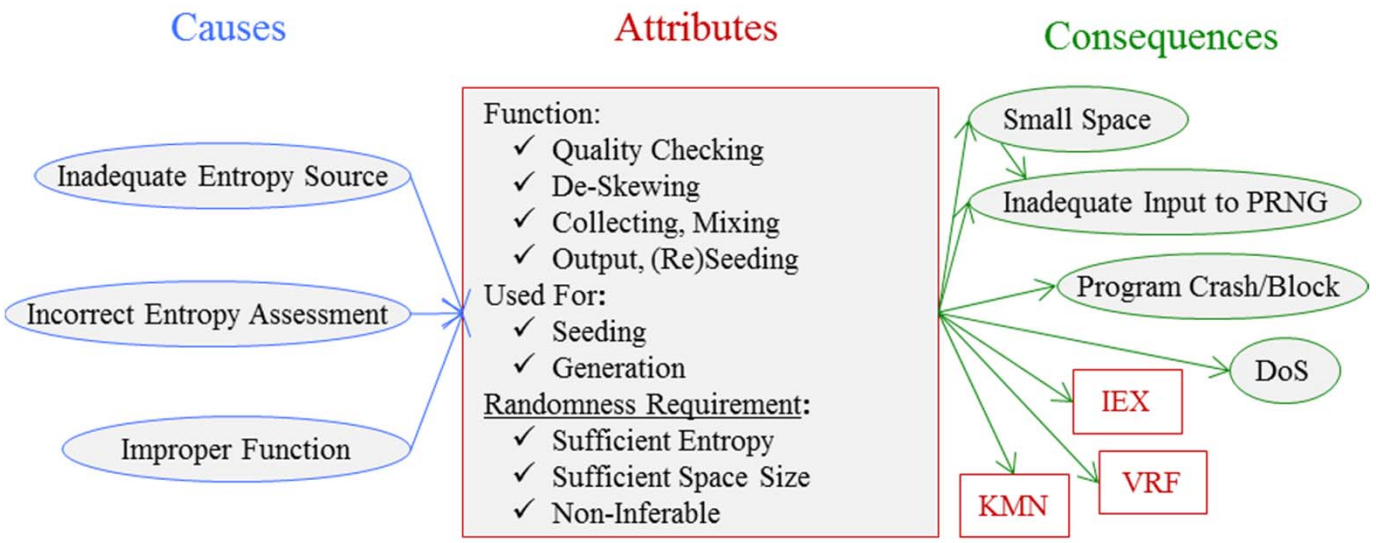
Randomness Bugs (RND)

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

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

- Related CWEs, SFPs and ST:
  - ✓ CWEs related to RND are CWE-330, 331, 332, 333, 334, 337, 339, 340, 341, 342, 343.
  - ✓ The related SFP cluster is SFP Primary Cluster: Predictability, which is CWE-905 with members CWE 330-344.

RND: Causes, Attributes, and Consequences



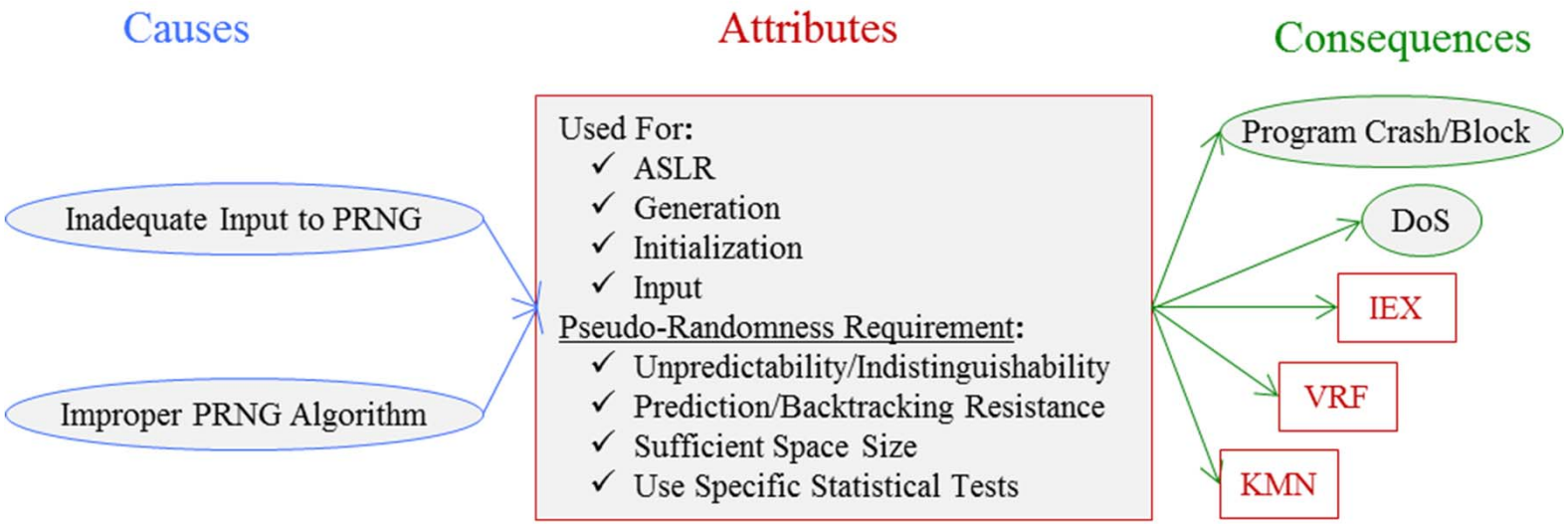
Pseudo-Randomness Bugs (PRN)

- Our Definition:
  The software generated output does not satisfy all use-specific pseudo-randomness requirements.

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

- Related CWEs, SFPs and ST:
  - ✓ CWEs related to PRN are CWE-330, 331, 332, 334, 335, 336, 337, 338, 339, 340, 341, 342, 343.
  - ✓ The related SFP cluster is SFP Primary Cluster: Predictability, which is CWE-905 with members CWE 330-344 [5].

PRN: Causes, Attributes, and Consequences



5. Future Work

Future Work

- One of our next steps is to explain more bugs using the developed BF classes.
- We also need to explore the use of ENC, VRF, and KMN in contexts other than cryptographic store and transfer. This will show where BF structures need refinements.
- Another step is to develop other BF classes. We are currently working on:
  - ✓ Authentication Bugs (ATN), Authorization Bugs (AUT)
  - ✓ Information Exposure (IEX)
  - ✓ Memory Allocation Bugs (MAL) – memory allocation faults (use after free, multiple free, failure to free)
  - ✓ Concurrency Bugs (CON)
  - + for Ockham needs, but may become BF classes:
    - Initialization Failure (INI)
    - Wrong Operation (WOP)
    - NULL or Incorrect Pointer Dereference (PTR)
    - Incorrect Arguments (ARG)

→ Our goal is for BF to become software developers' and testers' "*Best Friend*."

Questions

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