The Bugs Framework (BF) “Hands-On”

Tutorial – July 25, QRS 2017

Irena Bojanova

National Institute of Standards and Technology (NIST)

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

Outline

Session I. Enlightenment (40 min)

- Bugs Terminology
- Repositories of Bugs, Vulnerabilities, and Attacks
- Problems with Current Bug Descriptions
- Need for Structured Approach

Session II. The Bugs Framework (BF) (40 min)

- Context, Goal, Development and Evaluation
- Developed BF Classes: Definitions and Taxonomy
- Next BF Classes

Session III. “Hands-On” with Buffer Overflow and Injection (40 min)

- Buffer Overflow (BOF) Class
- Injection (INJ) Class

Session IV. “Hands-On” with Cryptography Bugs (45 min)

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

Session I. Enlightenment

- Bugs Terminology:
  - ✓ Software Weakness
  - ✓ Security Vulnerability
  - ✓ Software Attack
  - ✓ Security Failure
  - ✓ Source Code
- Repositories of Bugs, Vulnerabilities, and Attacks
  - ✓ Common Weakness Enumeration (CWE)
  - ✓ Software Fault Patterns (SFP)
  - ✓ Semantic Templates (ST)
  - ✓ NSA CAS Weakness Classes
  - ✓ Software State-of-the-Art Resources (SOAR) Matrix
  - ✓ SEI CERT C Coding Standard
  - ✓ Common Vulnerabilities and Exposures (CVE)
  - ✓ Open Web Application Security Project (OWASP): Vulnerability
  - ✓ Common Attack Pattern Enumeration and Classification (CAPEC)
- Problems with Current Bug Descriptions
- Need for Structured Approach

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

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)

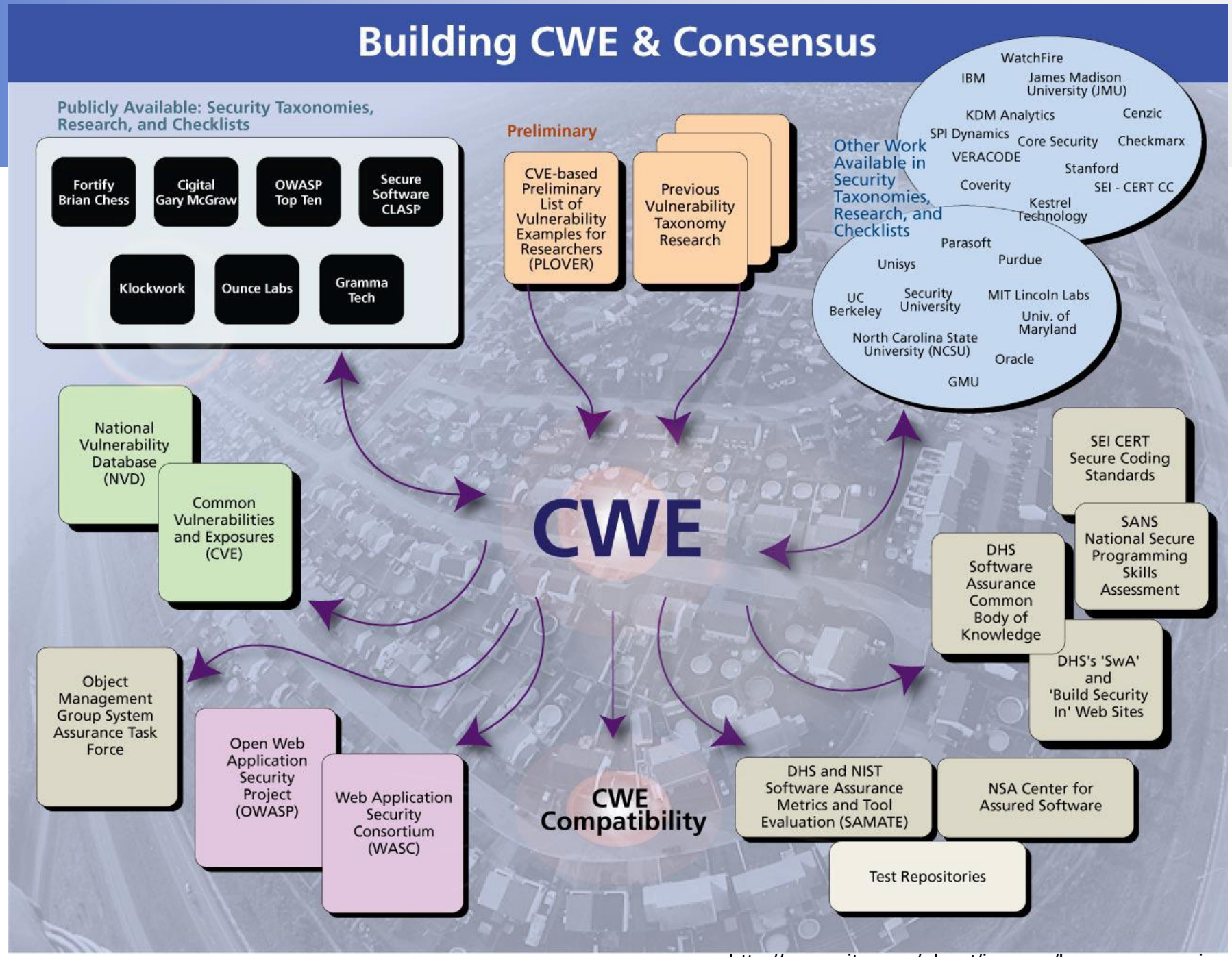
Common Weakness Enumeration (CWE)

- 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

CWE is a community effort.



http://cwe.mitre.org/about/images/lg_consensus.jpg

Fig.1. *CWE Efforts Context and Community [http://cwe.mitre.org*

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.

CWE Structure

- CWE is a collection of software weakness types stored as .xml, .xsd and .pdf documents.
- Major types of CWE-IDs:
  1. Category -- aggregates by types of weaknesses – Ex: CWE-355: User Interface Security Issues
  2. Compound Element – aggregates by group of events – Ex: CWE-476: NULL Pointer Dereference
  3. View – predefined perspectives – Ex: CWE-1000: Research Concepts
  4. Weakness – the covered weakness – Ex: CWE-311: Missing Encryption of Sensitive Data

→ Category and Compound Element are aggregations of weaknesses:

- ✓ Category aggregates types of weaknesses
- ✓ Compound Element aggregates several events that together can result in a successful attack.

→ View IDs are assigned to predefined perspectives with which to look at weaknesses in CWE.

CWEs Information

For each CWE the following information is provided:

- ✓ ID/Name of weakness type – see as example CWE 119
- ✓ Description – of behavior of this weakness, of exploit of the weakness
- ✓ Alternate Terms for this weakness
- ✓ Time of Introduction
- ✓ Applicable Platforms – languages
- ✓ Common Consequences of the exploit
- ✓ Likelihood of Exploit of this weakness
- ✓ Detection Methods
- ✓ Demonstrative Examples – code for languages/architectures
- ✓ Observed Examples – CVEs for which this type of weakness exists
- ✓ Potential Mitigations
- ✓ Relationships - with other CWEs
- ✓ Affected Resources
- ✓ Taxonomy Mappings - with OWASP, CERT C/C++, WASC, SFP
- ✓ Related Attack Patterns - CAPECs
- ✓ References.

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.

Software Fault Patterns (SFP)

- 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)

- 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.
- 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.

Semantic Templates (ST)

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

See phrases in descriptions and common consequences of CWE-120, colored according to ST:

- Fault
- Resource/Location
- Weaknes
- Consequence

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

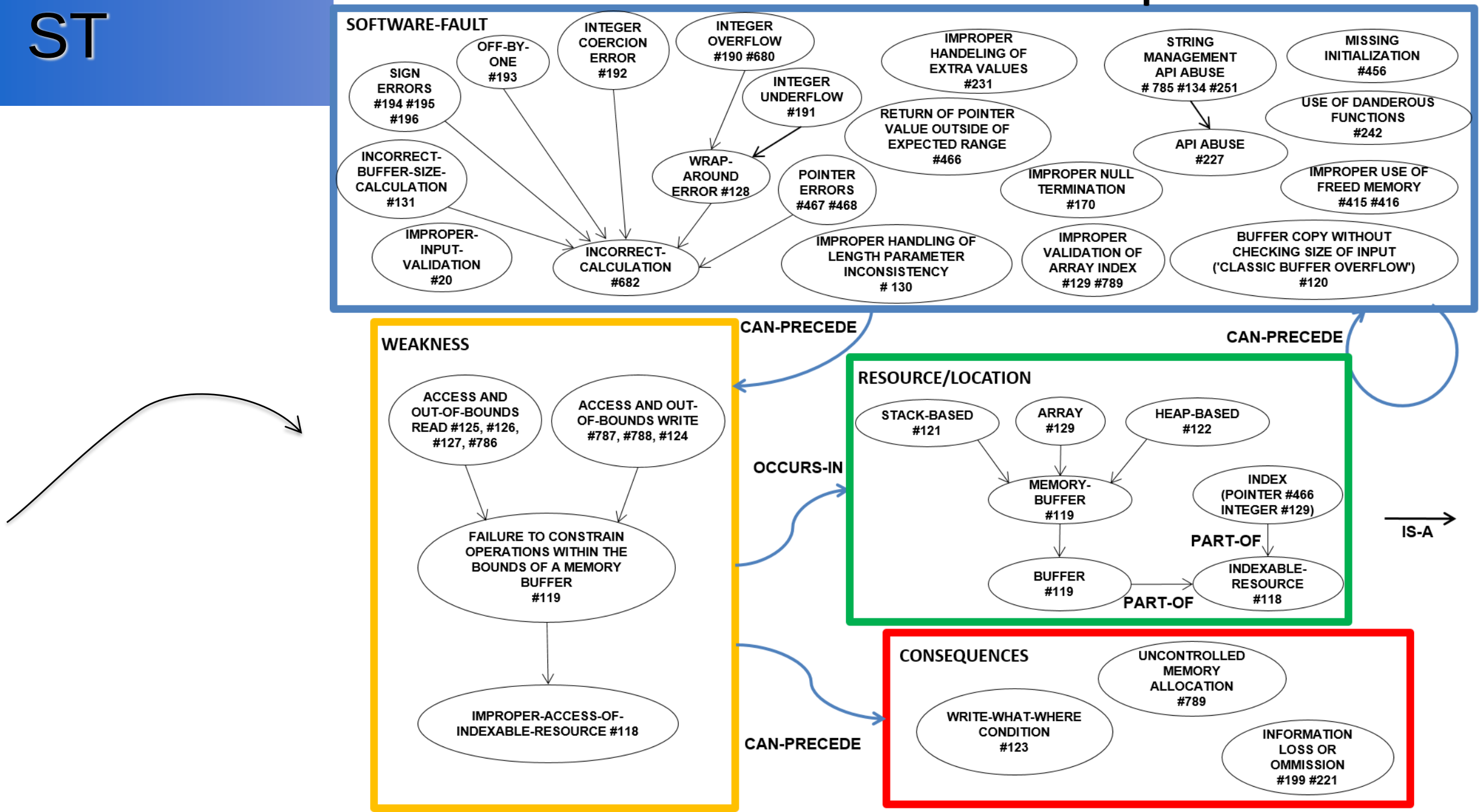
Description Summary: The program copies an input buffer to an output buffer without verifying that the size of the input buffer is less than the size of the output buffer, leading to a buffer overflow.

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.

ST

Buffer Overflow Semantic Template



NSA CAS Weakness Classes

The National Security Agency (NSA) Center for Assured Software (CAS) defines Weakness Classes in its "Static Analysis Tool Study - Methodology“.

→ See BF website.

Software State-of-the-Art Resources (SOAR) Matrix

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.

→ See BF website.

SEI CERT C Coding Standard

Software Engineering Institute (SEI), Carnegie Mellon University, CERT C Coding Standard

→ See BF website.

Common Vulnerabilities and Exposures (CVE)

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.

CVE's common identifiers:

- Enable data exchange between security products
- Provide baseline index point for evaluating coverage of tools and services.

→ See: https://cve.mitre.org/

Open Web Application Security Project (OWASP): Vulnerability

→ See BF website.

Common Attack Pattern Enumeration and Classification (CAPEC)

CAPEC) is a dictionary and classification taxonomy of known attacks

→ See: https://capec.mitre.org/

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 a Mess

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

CWE – Imprecise Definitions

- CWE-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.

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 instance, 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*

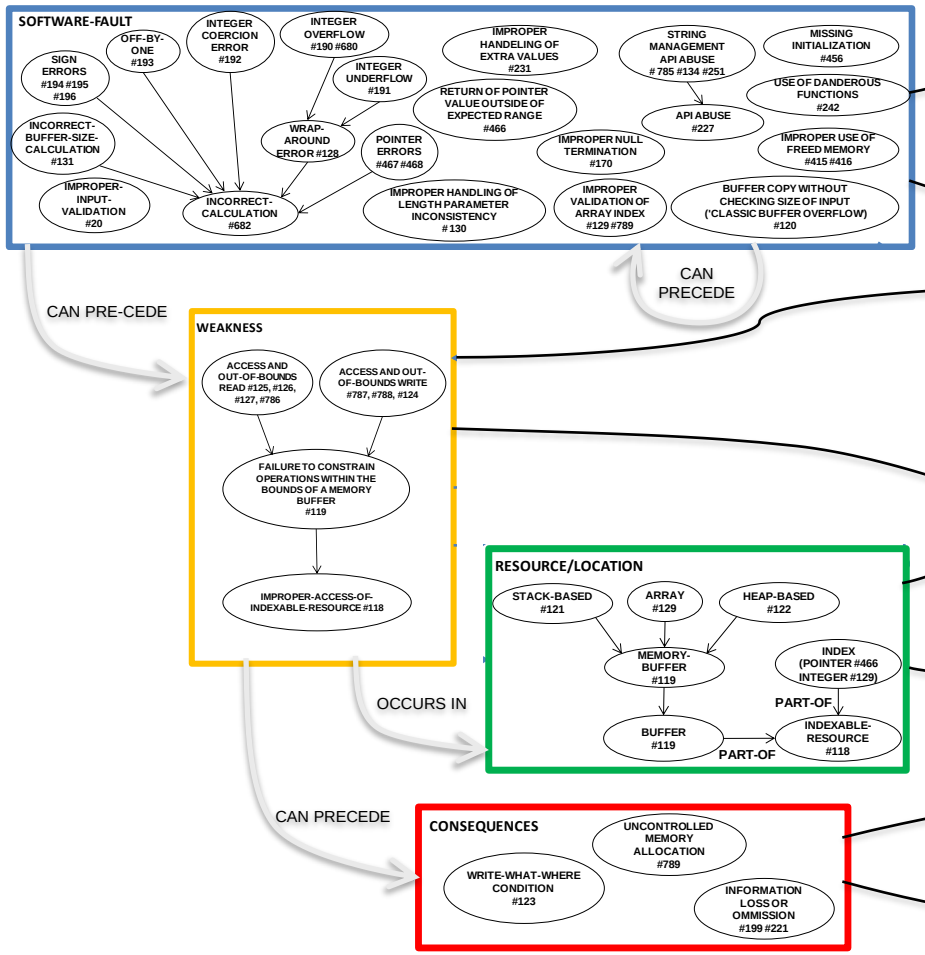
Summary: The software performs operations on a *memory buffer*, but it can *read from or write to a memory location that is outside of the intended boundary of the buffer*.
Extended description: Certain languages allow direct addressing of *memory locations* and *do not automatically ensure that these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

Other Bug Descriptions

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

→ 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).

Need for Structured Approach

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

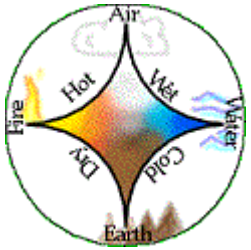
- As a result the newly delivered and the legacy systems will continue having security holes despite all the patching to correct errant behavior.

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

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

Periodic Table & Others to Describe Molecules

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



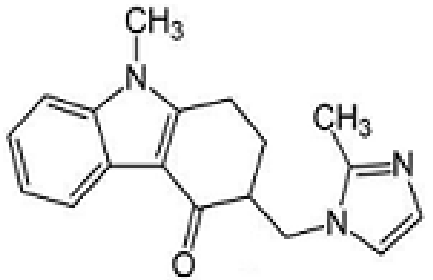
(Source: Reich Chemistry)

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

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

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

(Source: Wikimedia Commons)



$C_{18}H_{19}N_3O$

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

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

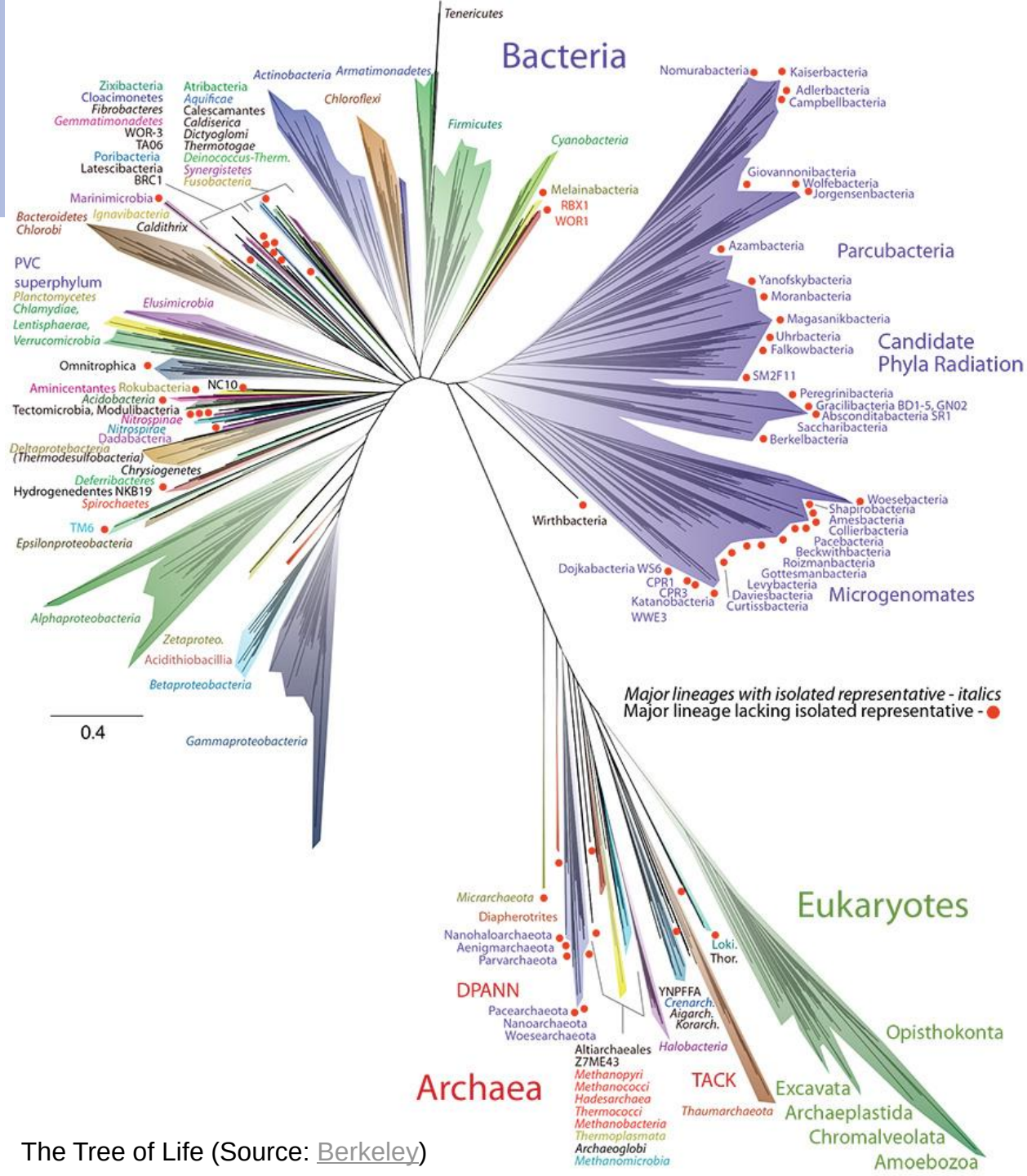
Columns correspond to the number of electrons in the outer shell and therefore the fundamental chemical properties.
Rows correspond to number of electron shells.

Tree of Life

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

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

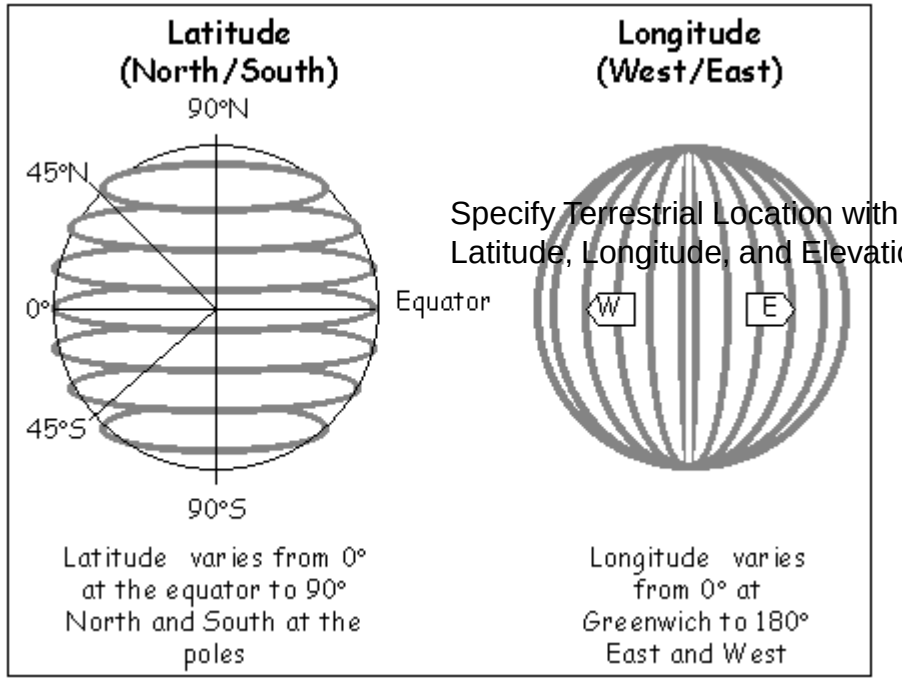
- Clearly shows "life we see around us – plants, animals, humans" and other Eukaryotes – represent a tiny percentage of world's biodiversity.

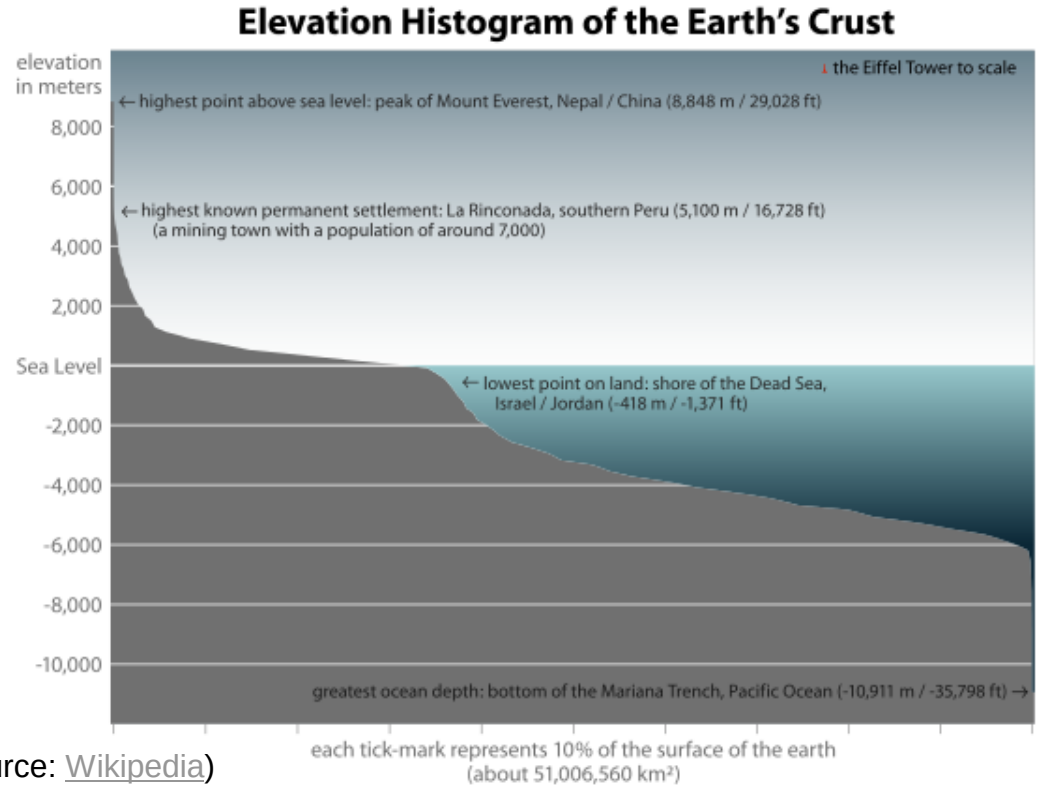


The Tree of Life (Source: Berkeley)

Geographic Coordinate System

Specify 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.

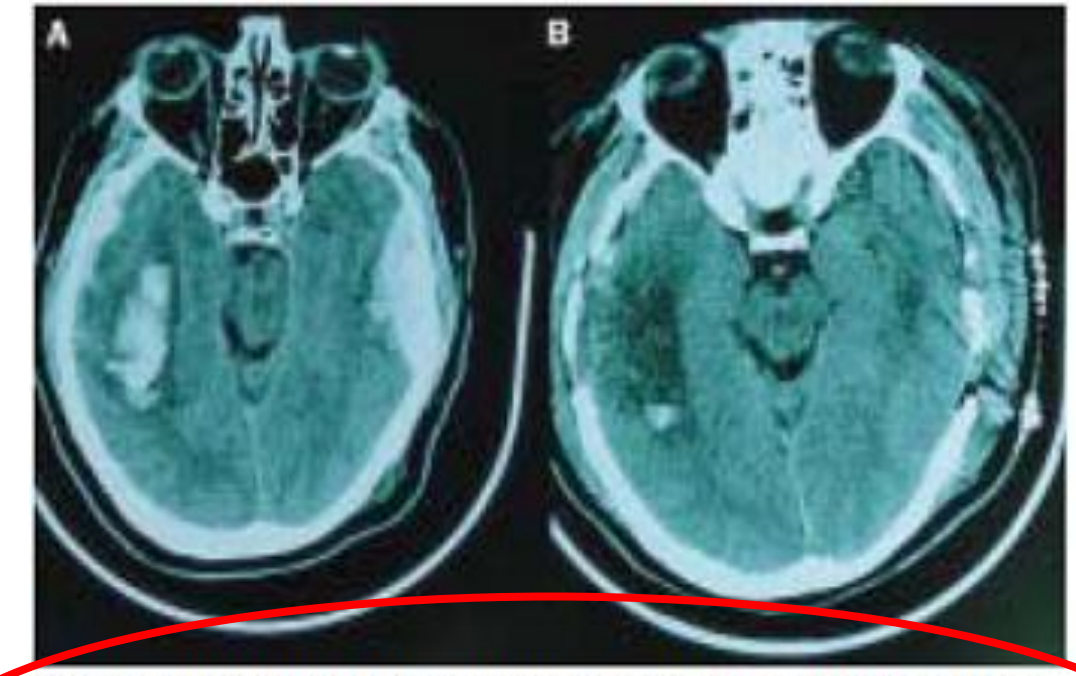


Figure 2: **Computed tomography of a comatose patient with a left temporal epidural haematoma, right parenchymal temporal lobe haematoma, and a right convexity subdural haematoma before and after craniotomy and evacuation of haematomas**

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

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

Session I – END

15 min BREAK

Session II starts at 2:25pm

Session II. The Bugs Framework (BF)

- Context, Goal, Development and Evaluation
- Developed BF Classes: Definitions and Taxonomy
  - ✓ Buffer Overflow (BOF)
  - ✓ Injection (INJ)
  - ✓ Control of Interaction Frequency Bugs (CIF)
  - ✓ Encryption Bugs (ENC)
  - ✓ Verification Bugs (VRF)
  - ✓ Key Management Bugs (KMN)
- Next BF Classes
  - ✓ Randomization Bugs (RND)
  - ✓ Authentication Bugs (ATN)
  - ✓ Authorization Bugs (AUT)
  - ✓ Information Exposure Bugs (IEX)
  - ✓ Faulty Operation (FOP)
  - ✓ Memory Allocation Bugs (MAL)

Context

- Currently there are no government, industry or international standards as to what constitutes the complete hierarchical classification of software bugs or a unified, formal approach for unambiguously expressing software vulnerabilities and chains of failures.
- Advances in scientific foundations of cybersecurity rely on the availability of:
  - ✓ accurate, precise, and unambiguous definitions of software weaknesses (bugs)
  - ✓ clear descriptions of software vulnerabilities.
- The myriad unprecedented attacks and security exposures, including on Internet of Things (IoT) applications, calls for serious efforts towards such formalization.

Goal

→ Create an unambiguous framework for expressing software bugs.

- Software developers, testers, and researchers need ways to:
  - ✓ Accurately and precisely comprehend and avoid all possible types of software bugs.
  - ✓ Develop techniques for making repeatable, quantified measurements of software quality and assurance.
  - ✓ Explain clearly applicability and utility of different software quality and assurance techniques or approaches.
  - ✓ Formally reason about assurance techniques or mitigation approaches that may work for a fault with certain attributes, but not for the same general kind of fault that has other attributes.
  - ✓ Estimate risk and determine best mitigation strategies based on known consequences of different kinds of faults.

The Bugs Framework (BF)

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

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.

What is the Bugs Framework (BF)?

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.

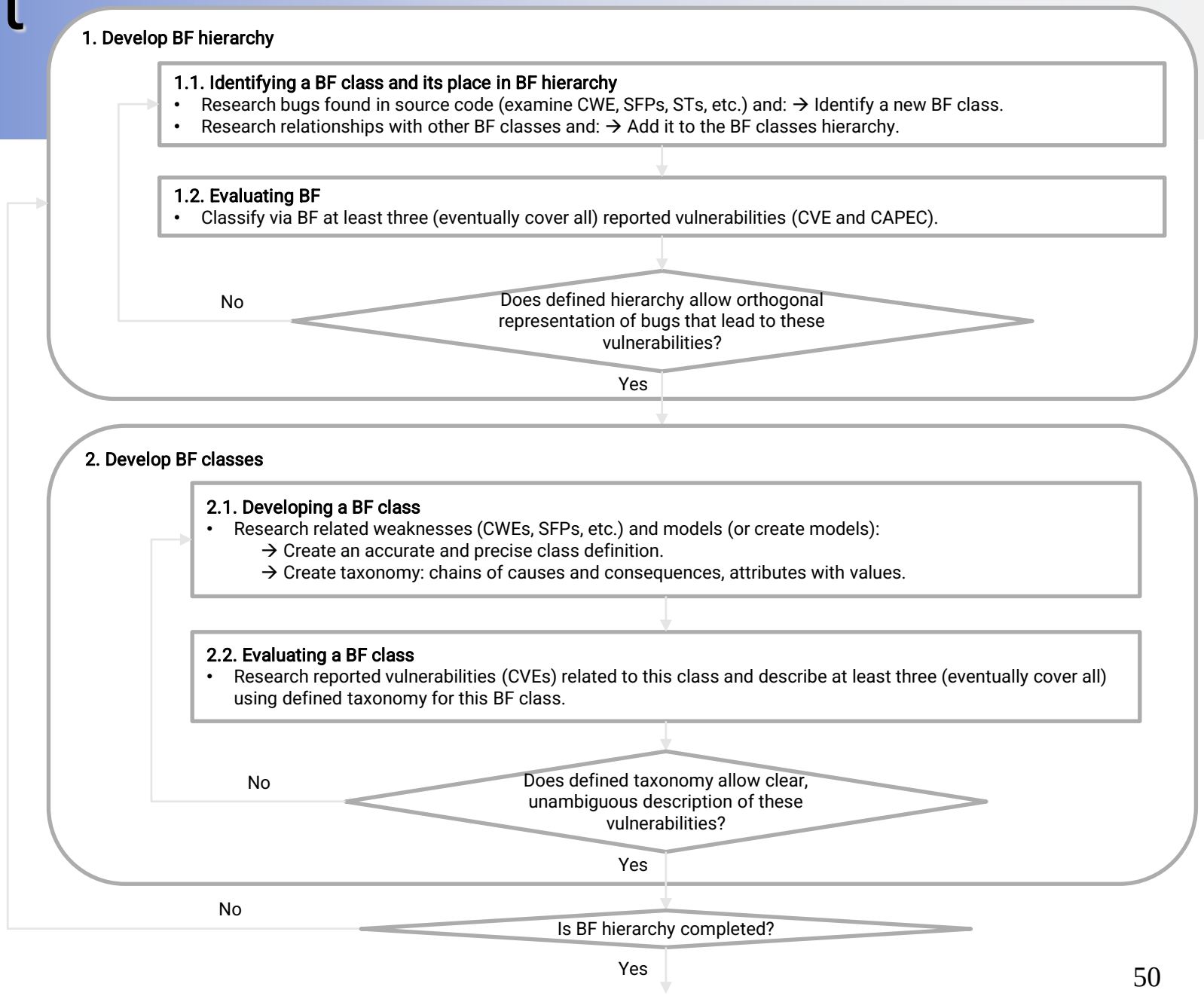
What is the Bugs Framework (BF)?

- 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: Questions

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

BF Development & Evaluation



BF Definitions and Descriptions Formats

Format of BF Class Definition:

The software does <<this and that wrong>>.

Format of BF Description of a Vulnerability:

<<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>>}.

, where: [] – "zero or one"
{} – "zero or more"

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)

Developed BF Classes

Buffer Overflow (BOF)

BF: Buffer Overflow (BOF)

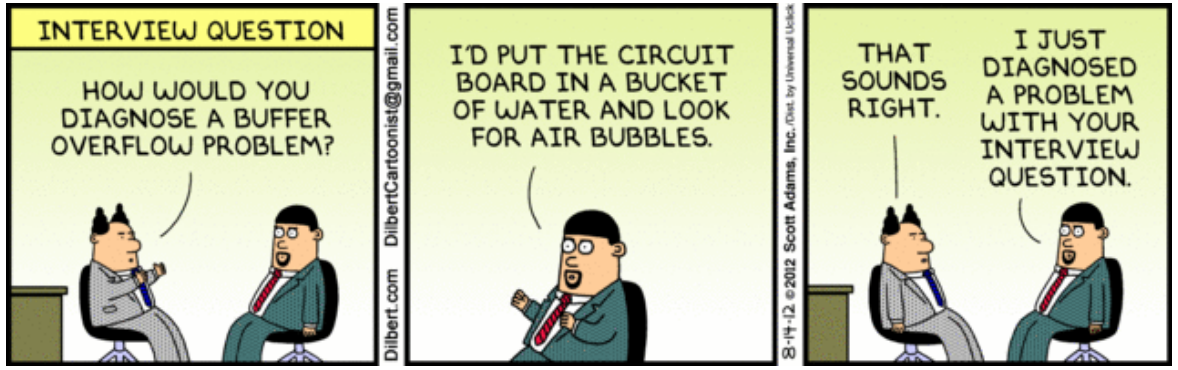
Buffer Overflow is the best class to begin with.

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

strcpy(destination, source);
```


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


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



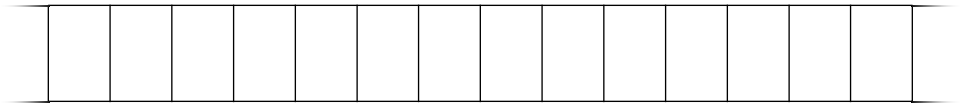
BF: Buffer Overflow (BOF)

- 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
  - ✓ Related 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.
  - ✓ The related SFP cluster is SFP8 Faulty Buffer Access under Primary Cluster: Memory Access.
  - ✓ The corresponding ST is the Buffer Overflow Semantic Template.

BOF Attributes

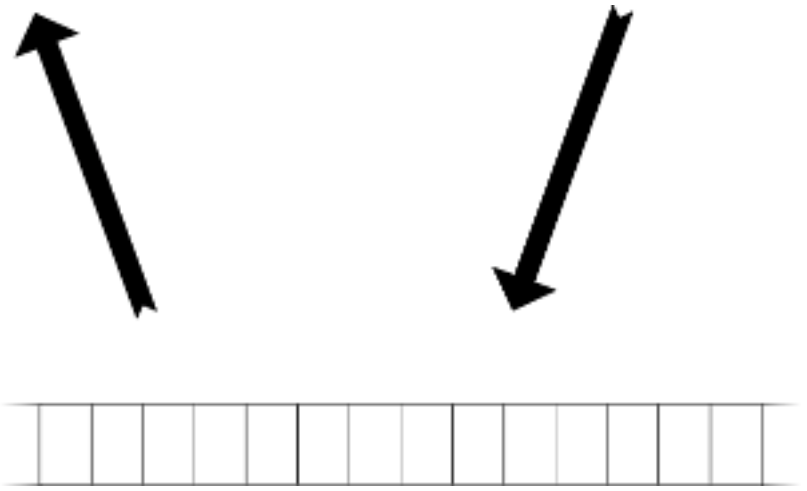
- Often referred to as a “buffer,” an array is a contiguously allocated set of objects, called elements.
  - ✓ Has a definite size – a definite number of elements are allocated to it.
  - ✓ Software should not use array name to access anything outside boundary of allocated elements.
  - ✓ Elements are 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.



✓ Note:

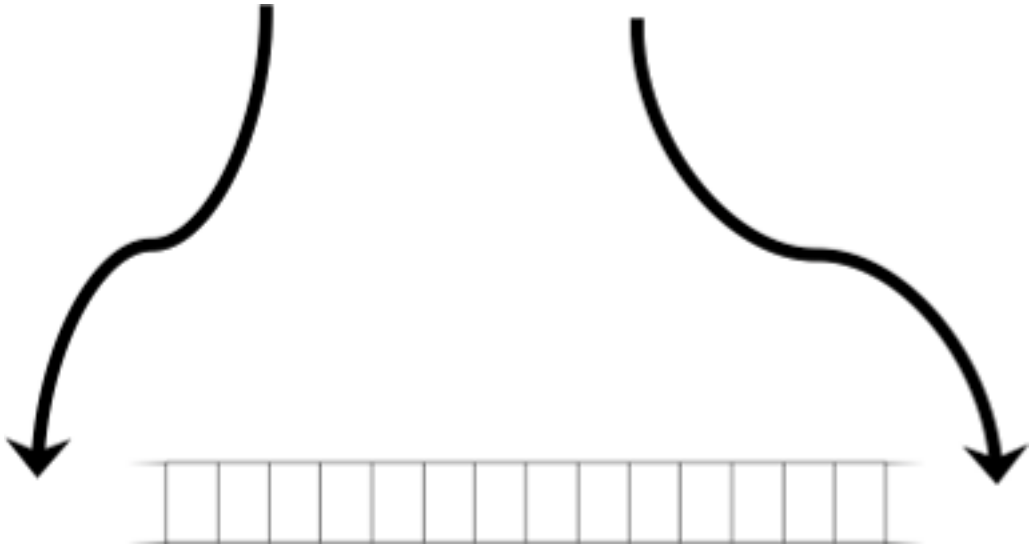
The underlined attribute shows what eventually goes wrong.

The rest of the attributes are simply descriptive.

BOF Attributes – Boundary

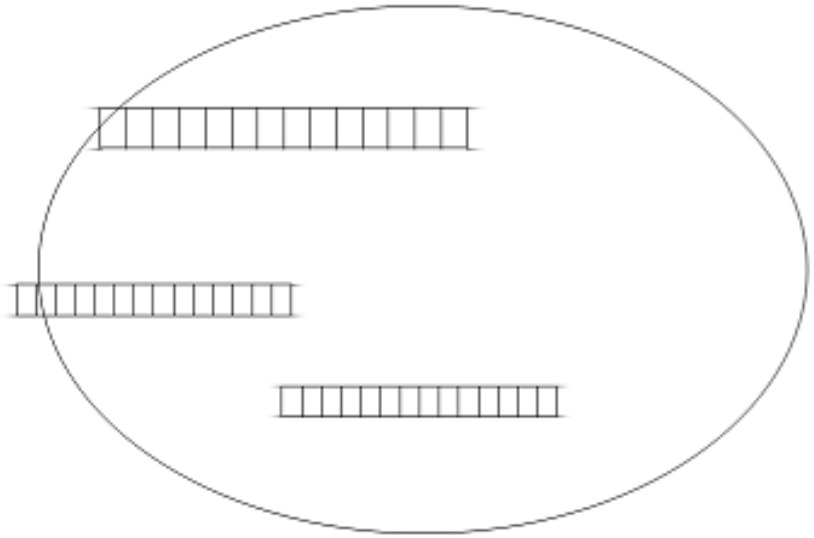
- Access: Read, Write.
- **Boundary** – indicates which end of the array is violated:
  - ✓ Below (before, under, or lower)
  - ✓ Above (after, over, or upper).

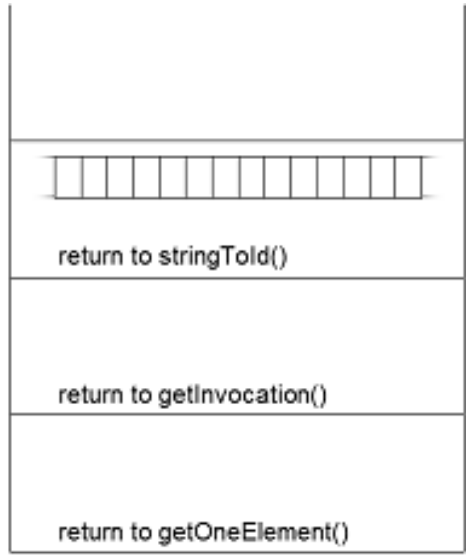
- Synonyms for boundary are side or bound.
- Before, under or lower may be used instead of below.
- After, over or upper may be used instead of above.
- Outside indicates boundary is unknown or it doesn't matter.



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).

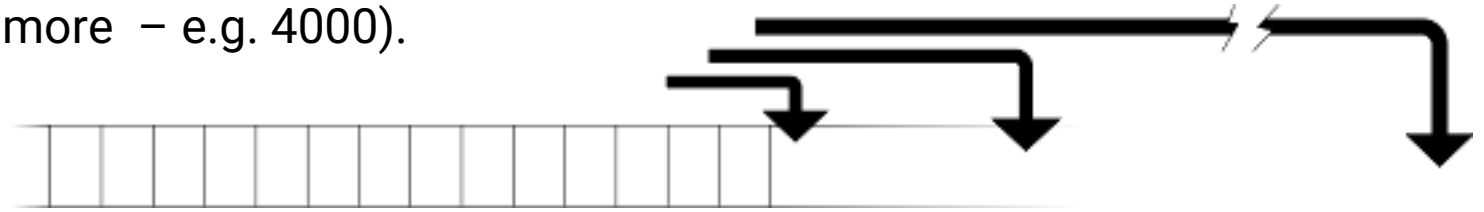




- It may matter since:
  - violations in the stack may affect program execution flow
  - while violations in the heap typically only affect data values.
- Other compilers and operating system may have other locations that are significant
  - e.g. BSS, Data, Code (text).

BOF Attributes – Magnitude

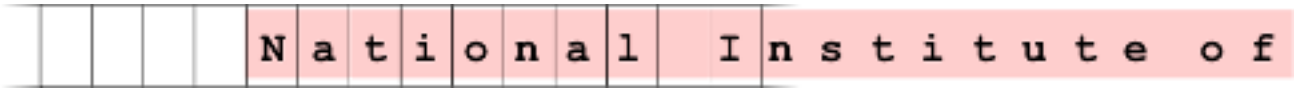
- Access: Read, Write.
- Boundary: Below, Above.
- Location: Heap, Stack, BSS, Data, Code.
- **Magnitude** – how far outside the boundary the violation extends:
  - ✓ Small (just barely outside – e.g. one to a few bytes)
  - ✓ Moderate (from 8 to dozens of bites)
  - ✓ Far (hundreds, thousands or more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

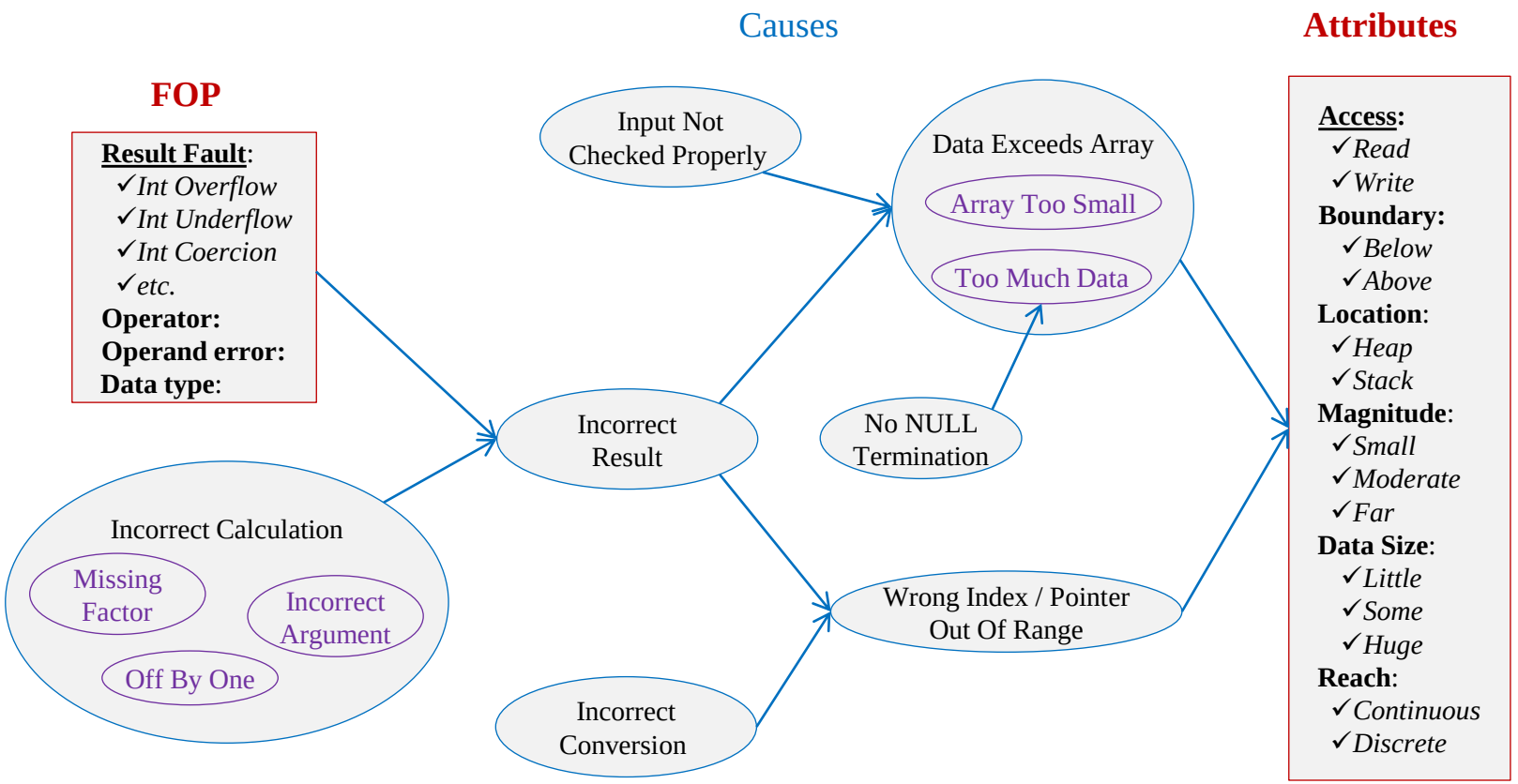
- Access: Read, Write.
- Boundary: Below, Above.
- Location: Heap, Stack, BSS, Data, Code.
- Magnitude: Small, Moderate, Far.
- Data Size: Little, Some, Huge.
- **Reach** – one-by-one or arbitrary:
  - ✓ Continuous
  - ✓ Discrete.

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 reach 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



Causes:

- Only two proximate causes of BOF:
  - ✓ Data Exceeds Array
  - ✓ Wrong Index / Pointer Out of Range.

Attributes:

- Underlined – shows what eventually goes wrong.
- The rest – simply descriptive.

BOF: Causes

- Only two proximate causes of buffer overflows:
  - ✓ “Data Exceeds Array” – amount of data exceeds the array size
  - ✓ “Wrong Index/ Pinter Out Of Range” there is a wrong index or pointer.
- Preceding causes may lead to them.

  “Data Exceeds Array” could be the result of:
  - ✓ “Array Too Small” – incorrectly small array is allocated, so destination is too small – may occur because programmer:
    - leaves out a factor, like the size of a header
    - uses wrong variable
    - forgets room for a null to terminate a string.
  - ✓ “Too Much Data” – incorrectly large amount of data is accessed, so data is too big – may occur because:
    - the string is not NULL terminated
    - the amount of data is calculated differently than the size of the buffer (e.g. heartbleed).

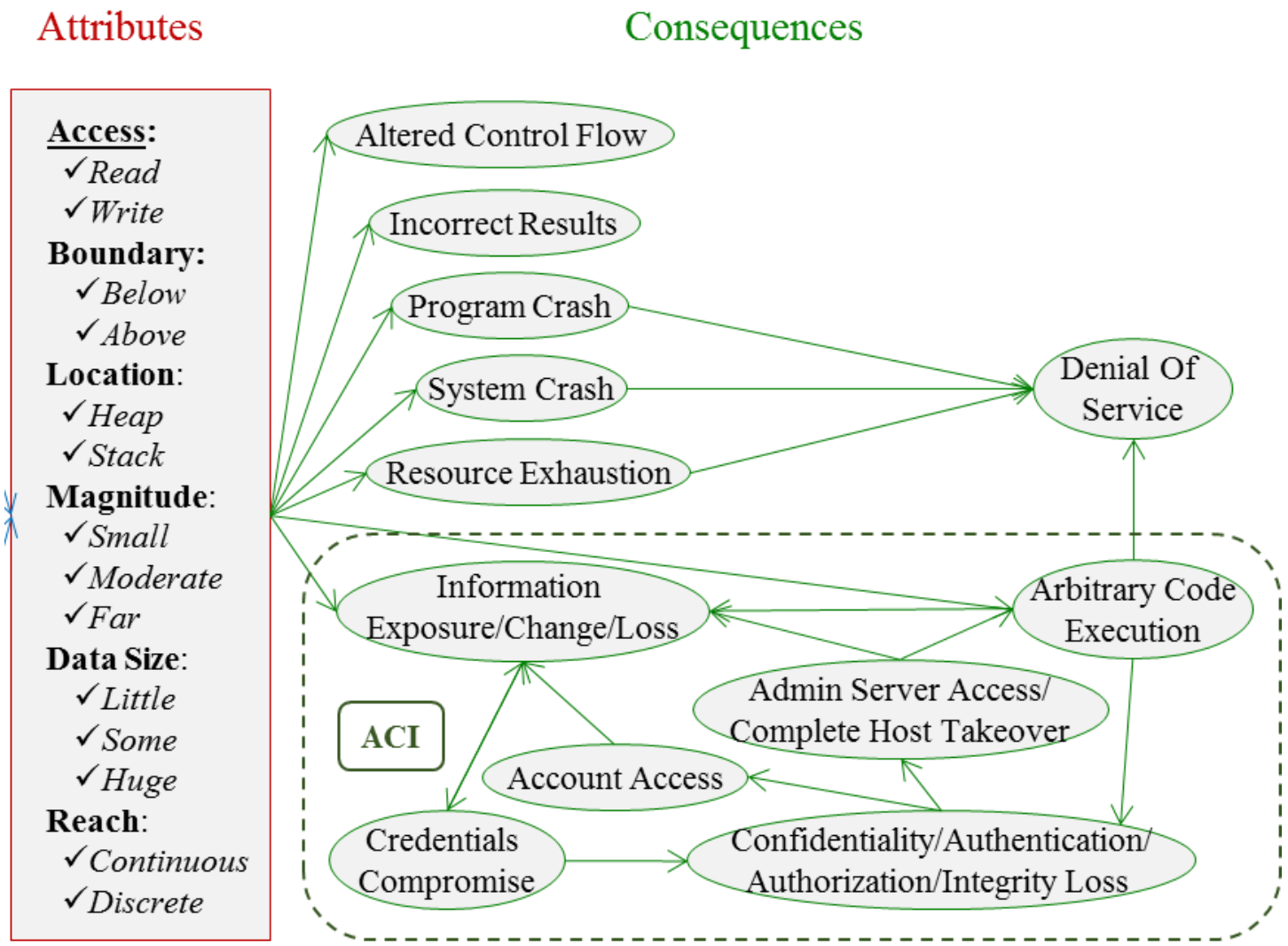
BOF: Causes – Exposition

When we examine code, we can say:

- In some cases, the programmer allocated the array too small,
  such as in CVE-2015-0235 – Ghost.
  The code computes the size of a buffer needed,
  but leaves out one factor,
  which makes the buffer four bytes short.
- In other cases, too much data was accessed,
  such as in CVE-2014-0160 – Heartbleed.
  A string was stored in an array,
  but instead of computing the length of the string,
  the code used a length from the input,
  which was not checked against the string.
  This can cause the code to read far more from the buffer than was allocated.

In both cases, size of data exceeds size of array. Just looking at code, it may be difficult to determine which case it is – it needs semantic content. That's why the two are sub-causes of one cause.

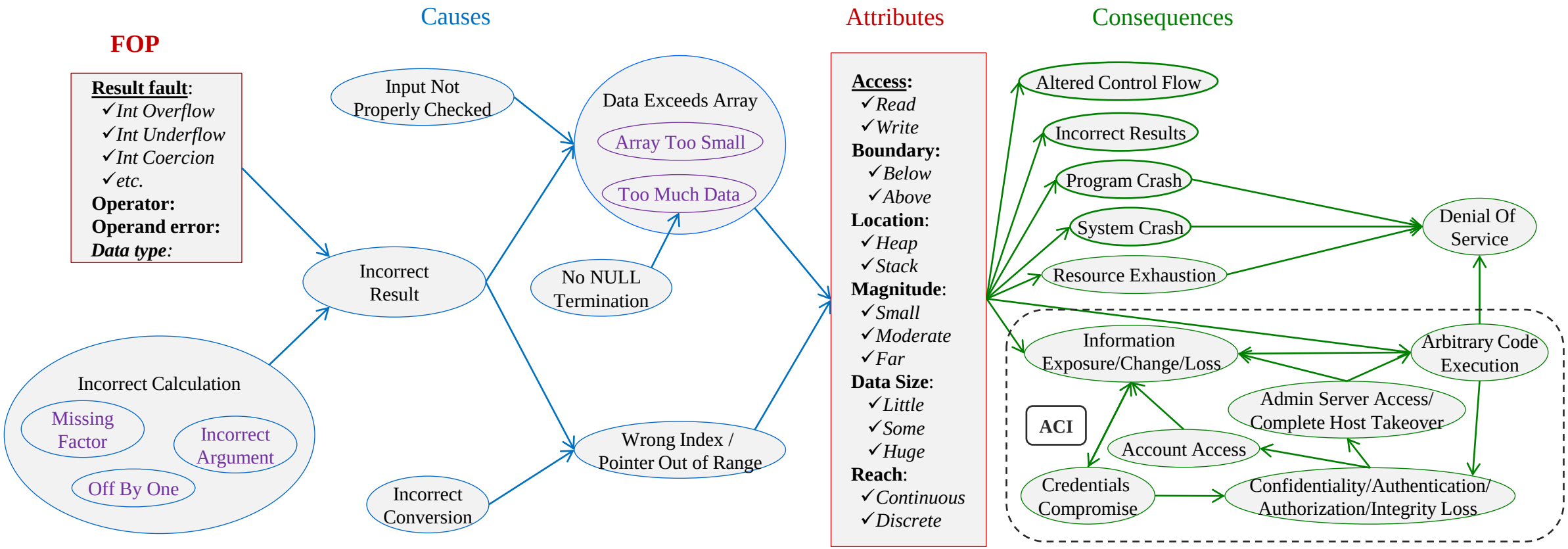
BOF: Consequences



Shows what could happen due to the fault.

Note: "Resource Exhaustion" refers to Memory and CPU.

BOF: Causes, Attributes, and Consequences



BOF: Sites

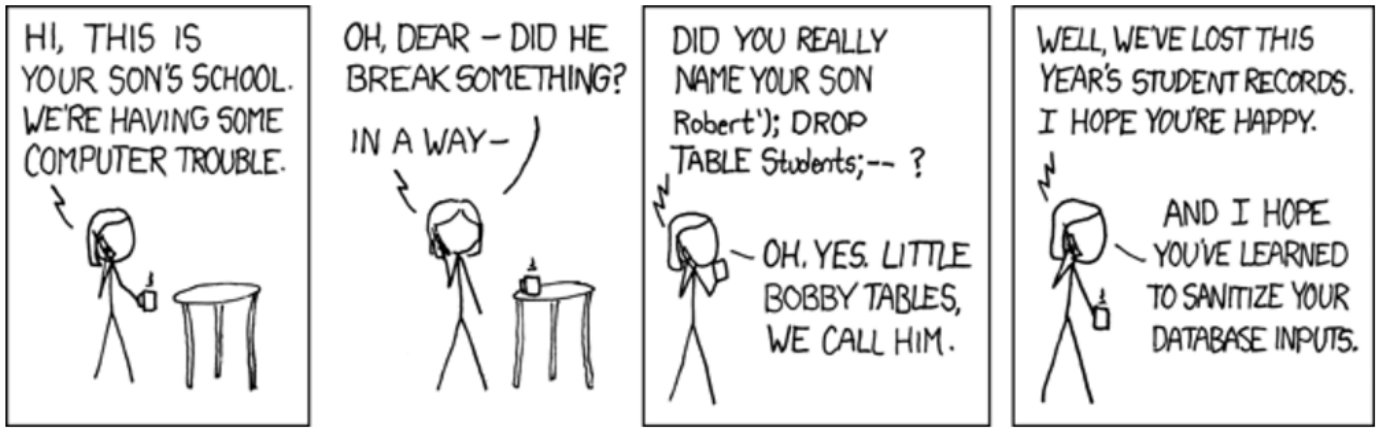
- In C, Buffer Overflow may occur at:
  - ✓ Use of [ ] operator with arrays in C
  - ✓ Use of unary * operator with arrays in C
  - ✓ Use of string library functions, such as `strcpy()` or `strcat()`.

Developed BF Classes

Injection (INJ)

BF: Injection (INJ)

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



Source: xkcd

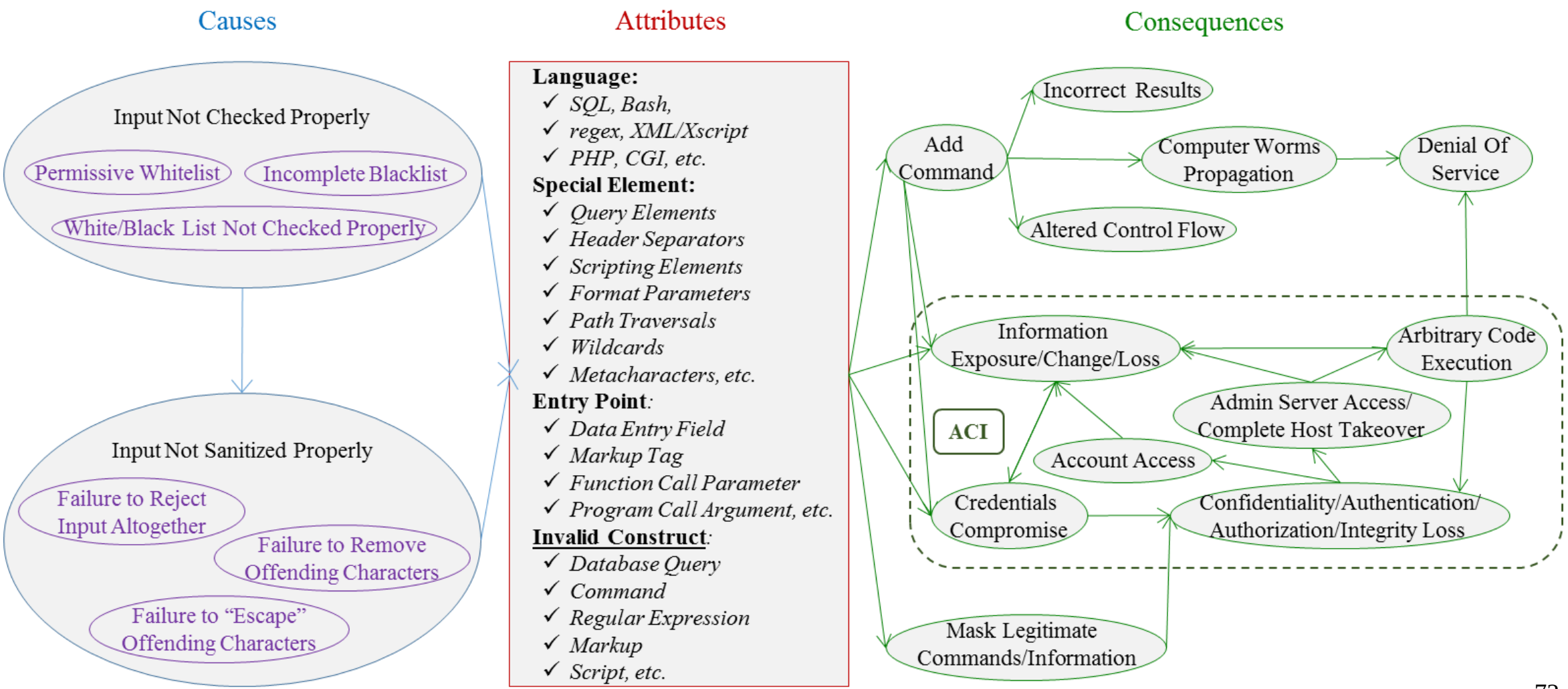
BF: Injection (INJ)

- 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.

- 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: Causes

- Input Not Checked Properly means:
  - not checked at all or
  - the check was not the right check.
- Improper Sanitization implies that:
  - proper check was done,
    but the sanitization is not sufficient.
- Failure to Reject Input Altogether –
  e.g. ”Input not allowed, please try again“.

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".)

INJ: Consequences

- Examples of immediate consequences:
  - ✓ Add Command:
    turn
    ```
    touch file
    ```
    into
    ```
    touch file; rm /etc/passwd
    ```
  - ✓ Mask Legitimate Commands or Information:
    turn
    ```
    WHERE username='name' AND password='psw'
    ```
    into
    ```
    WHERE username='name' AND 1=1 --password='psw'
    ```
    so that the check for password is skipped.

Developed BF Classes

Control of Interaction Frequency (CIF)

BF: Control of Interaction Frequency (CIF)

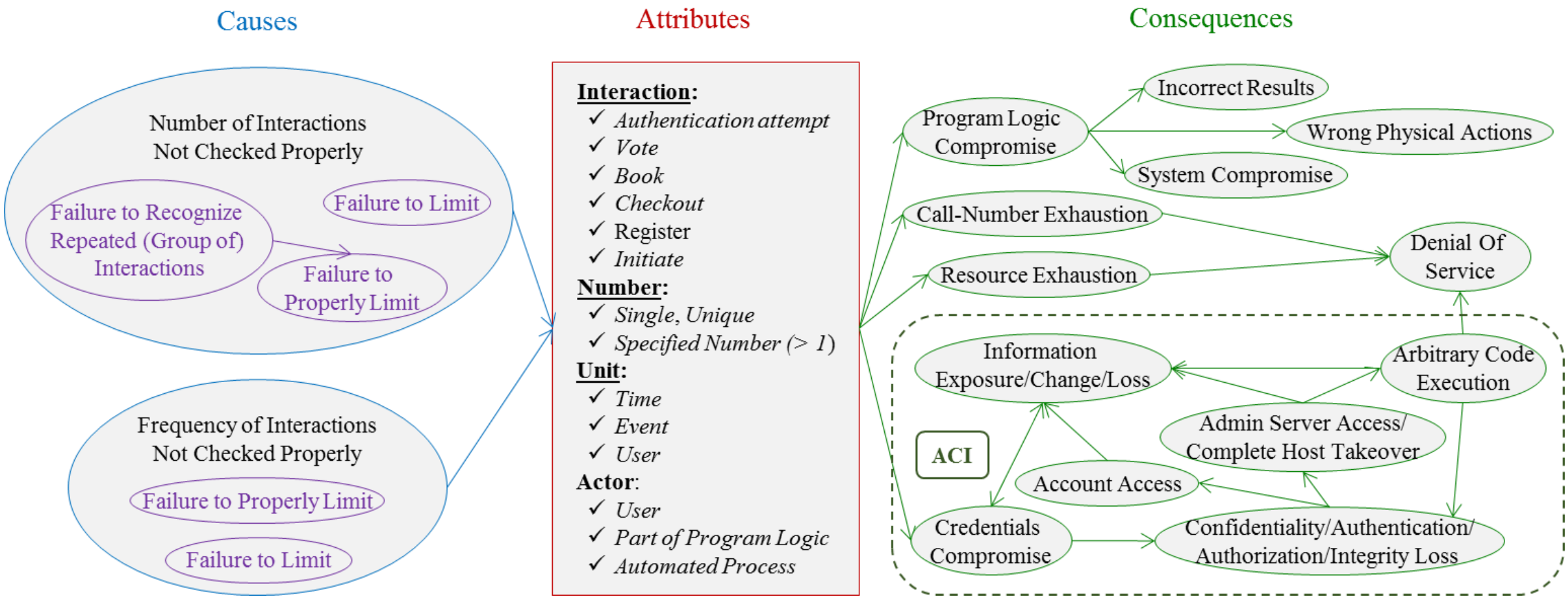
- Our Definition:
  The software does not properly limit the number of repeating interactions per specified unit.

  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 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.

CIF: Consequences

“Credentials” concerns:

- ✓ username or password
- ✓ smart card and personal identification number (PIN)
- ✓ retina, iris, fingerprint, etc.

“Resource Exhaustion” concerns:

- ✓ Memory
- ✓ CPU
- ✓ Granted licenses.

Developed BF Classes

Cryptography Classes:

- 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.

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 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

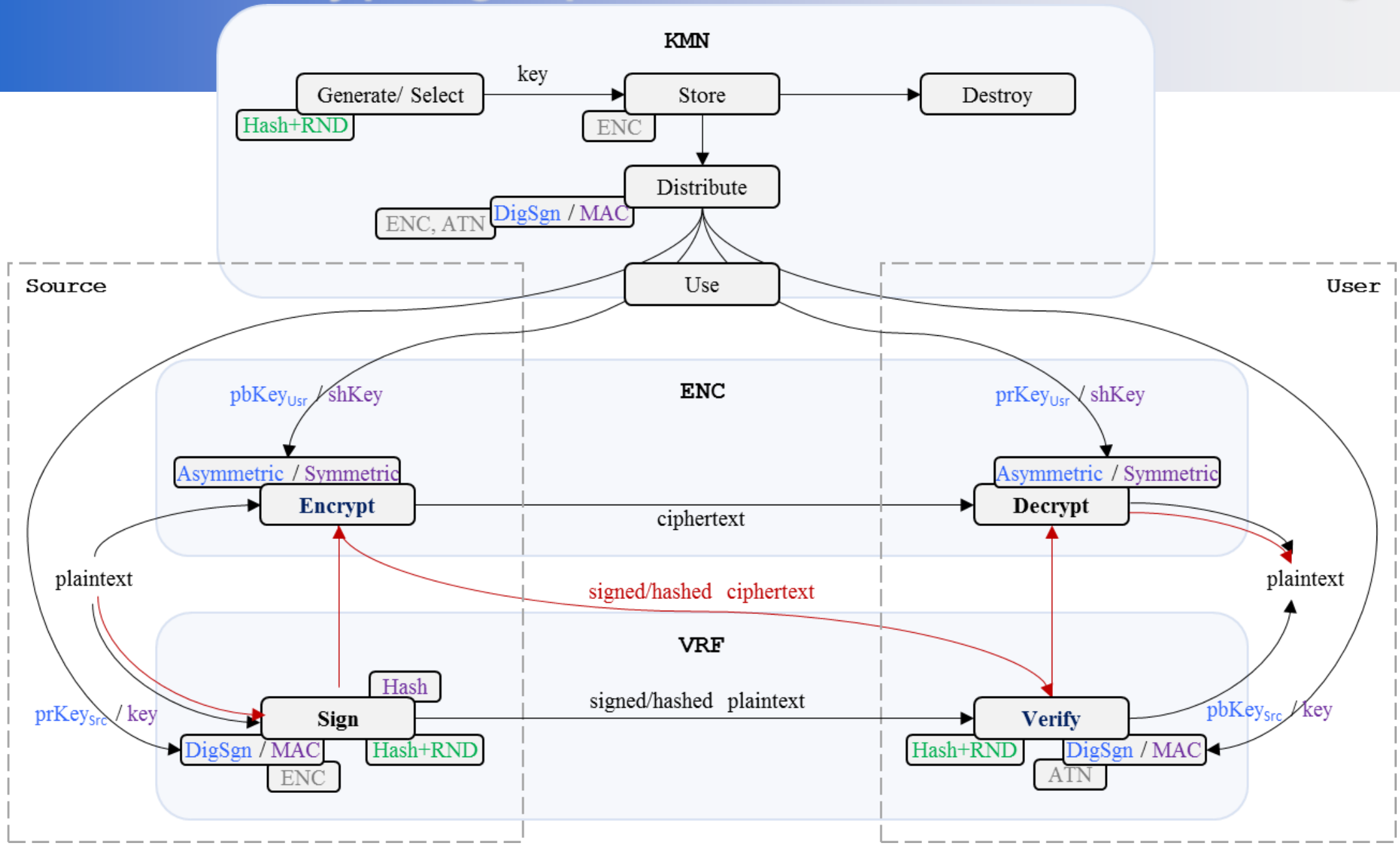
A modern, secure, flexible cryptographic storage or transfer protocol:

- 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.

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

Model of Cryptographic Store or Transfer Bugs



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.

- 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 $pbKey_{Usr}$ and signs with $prKey_{Src}$.
  - ✓ User decrypts with $prKey_{Usr}$ and verifies with $pbKey_{Src}$.

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: 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 $pbKey_{Src}$.

Developed BF Classes

Encryption Bugs (ENC)

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:

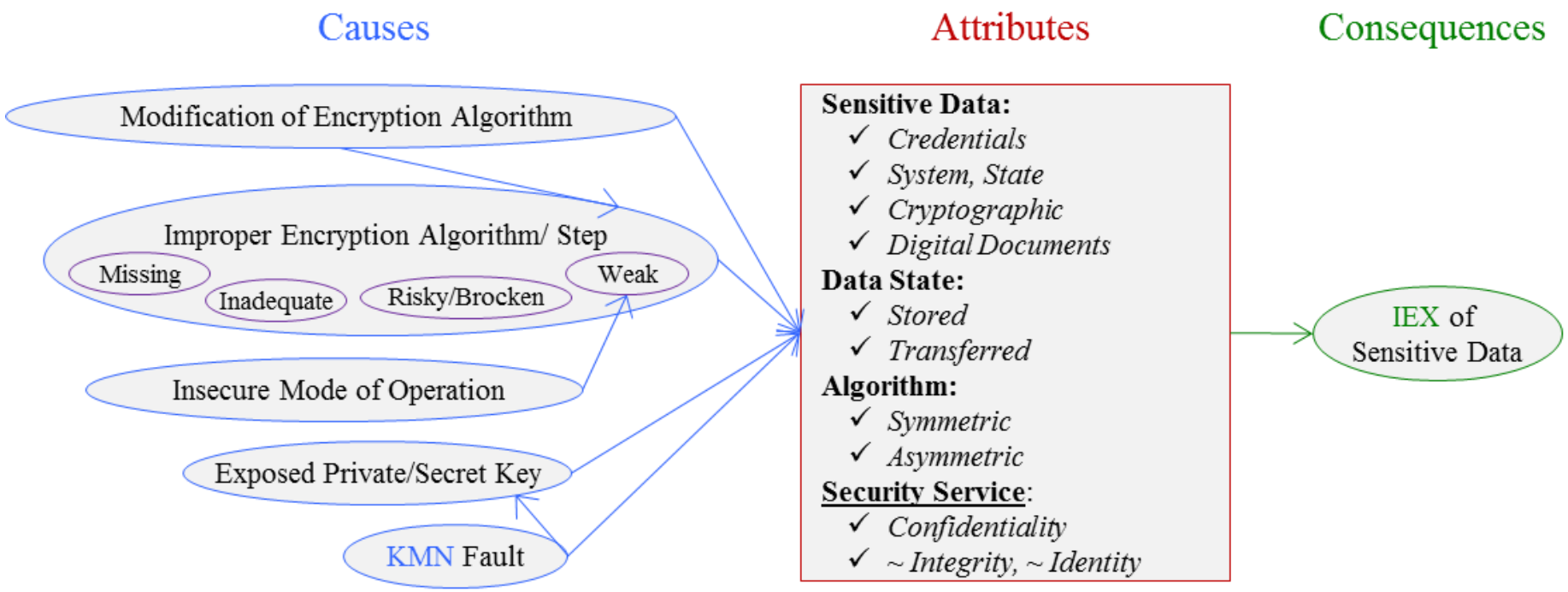
  The software does not properly transform ciphertext into plaintext using cryptographic algorithm and key(s).

  Note that "transform" is for confidentiality.

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

- Related CWEs, SFPs and ST:
  - ✓ CWEs related to ENC are CWE-256, 257, 261, 311-318, 325, 326, 327, 329, 780.
  - ✓ Related SFP clusters are 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.

Developed BF Classes

Verification Bugs (VRF)

BF: Verification Bugs (VRF)

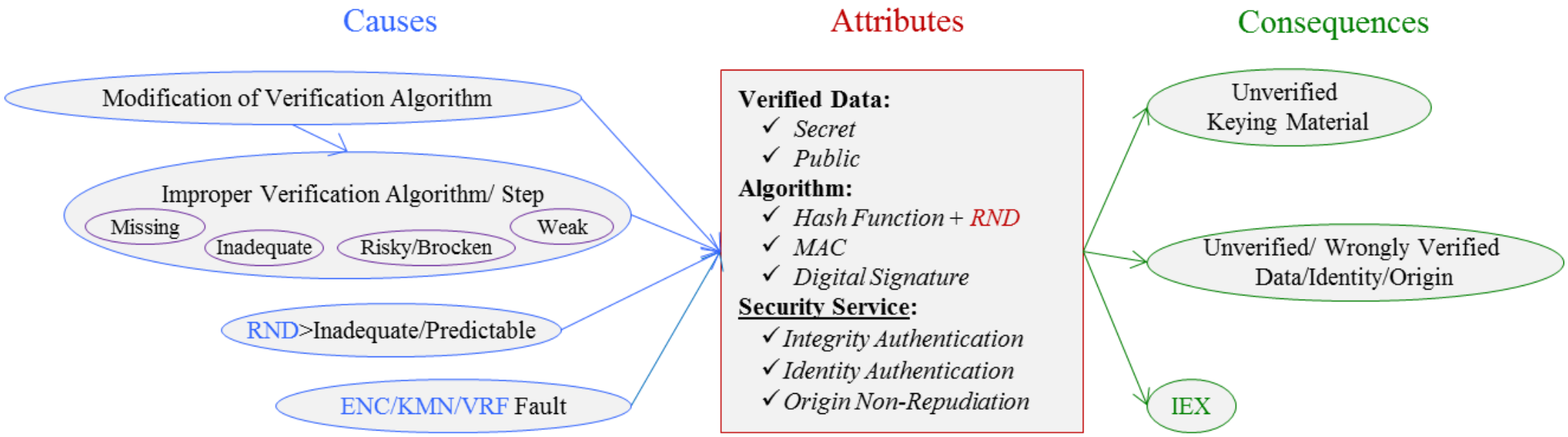
- Our Definition:
  The software does not properly sign data, check and prove source, or assure data is not altered.

  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, RND, ENC, Authentication (ATN), IEX.

- Related CWEs, SFPs and ST:
  - ✓ CWEs related to VRF are CWE-295, 296, 347.
  - ✓ Related SFP cluster is 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.
- **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.

Developed BF Classes

Key Management Bugs (KMN)

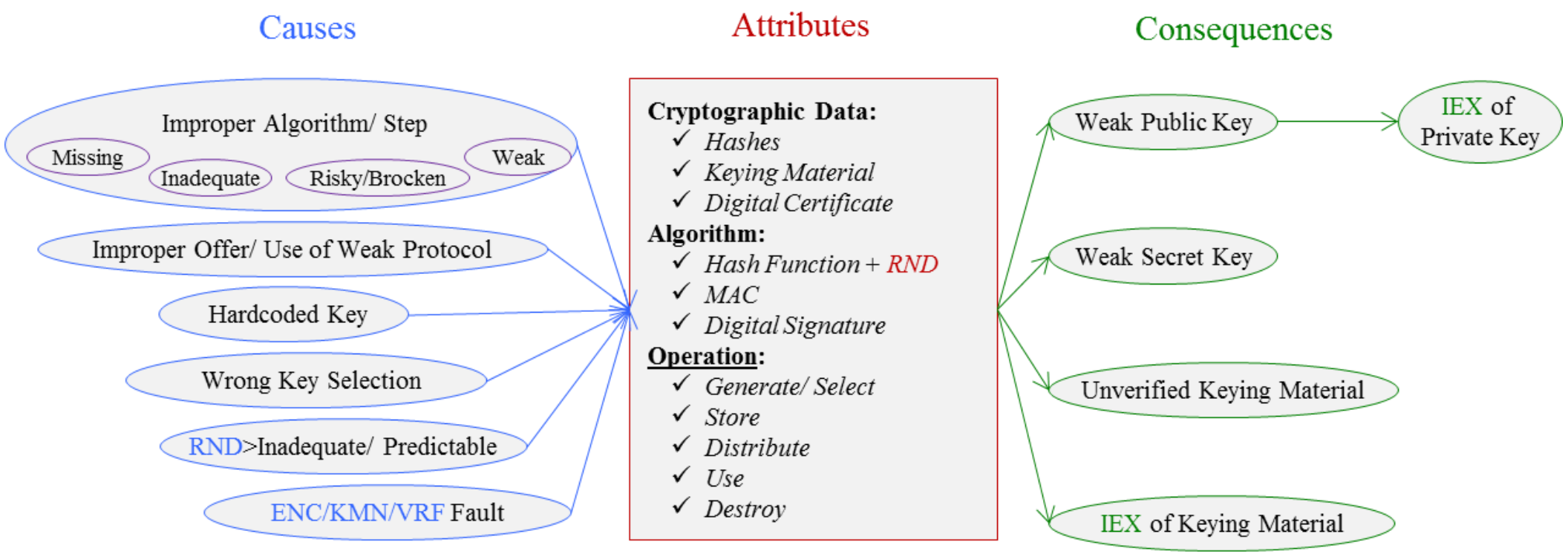
BF: Key Management Bugs (KMN)

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

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

- Related CWEs, SFPs and ST:
  - ✓ CWEs related to KMN are CWE-321, 322, 323, 324.
  - ✓ Related SFP clusters are 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.
- **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.

BF: Next Steps

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

Upcoming: BF Access Classes

- BF Classes related to Access:
  - ✓ ATN (Authentication Bugs)
  - ✓ AUT (Authorization – often conflated with Access Control)
  - ✓ ENC (Encryption Bugs)
  - ✓ VRF (Verification)
  - ✓ KMN (Key Management)
  - ✓ RND (Randomization Bugs)
  - ✓ CIF (Control of Interaction Frequency Bugs)
  - ✓ IEX (Information Exposure).

Upcoming: BF Functionality Classes

- BF Classes related to Functionality:
  - ✓ FOP (Faulty Operation) – Calculations, Comparisons, Functions, Cast
    Integer Overflow, Divide by Zero, …
  - ✓ FLO (Control Flow Bugs) – Branching, Looping (Switch without default, Infinite loop,...)
  - ✓ INJ (Injection)
  - ✓ EXC (Exception Handling Bugs)
    – Throw, Try, Catch
  - ✓ CON (Concurrency Bugs)
    – Deadlock, Starvation (unfair scheduling), Races, Locks, Synchronization, etc.

Upcoming: BF Data Classes

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

Session II – END

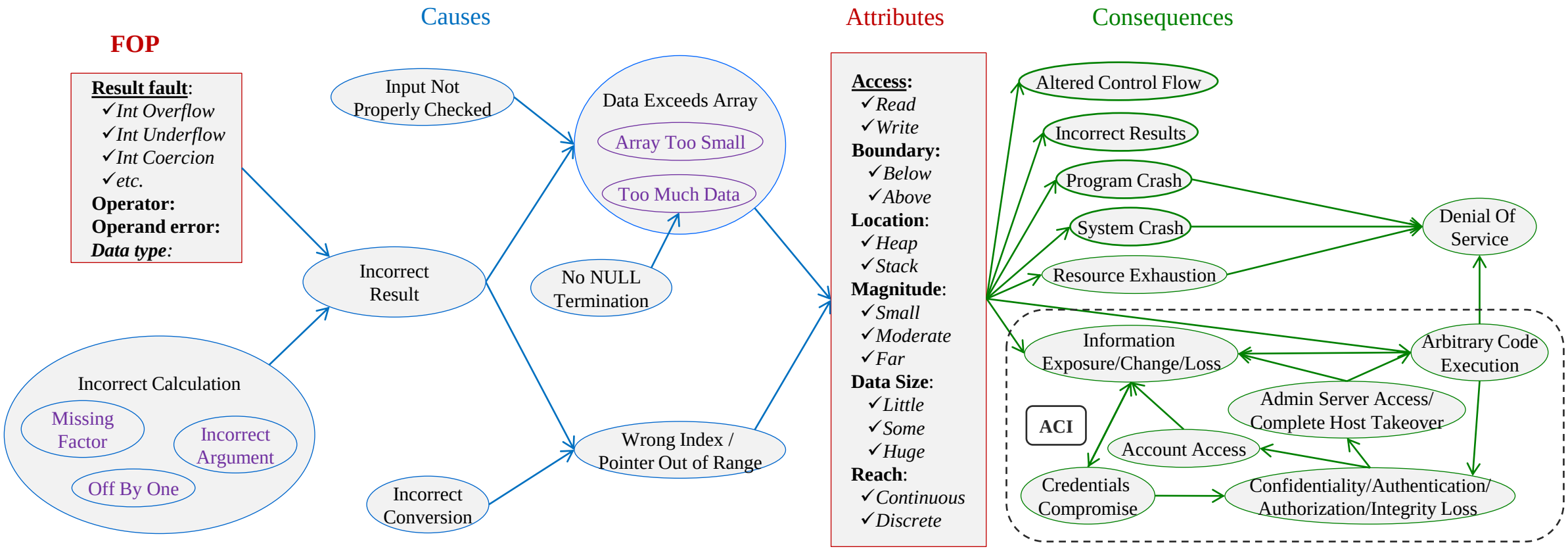
15 min BREAK

Session III starts at 3:20pm

Session III. “Hands-On” with Buffer Overflow and Injection

- Buffer Overflow (BOF) Class
  - ✓ Examples of Applying BOF
  - ✓ Exercises on Applying BOF
- Injection (INJ) Class
  - ✓ Examples on Applying INJ
  - ✓ Exercises on Applying INJ

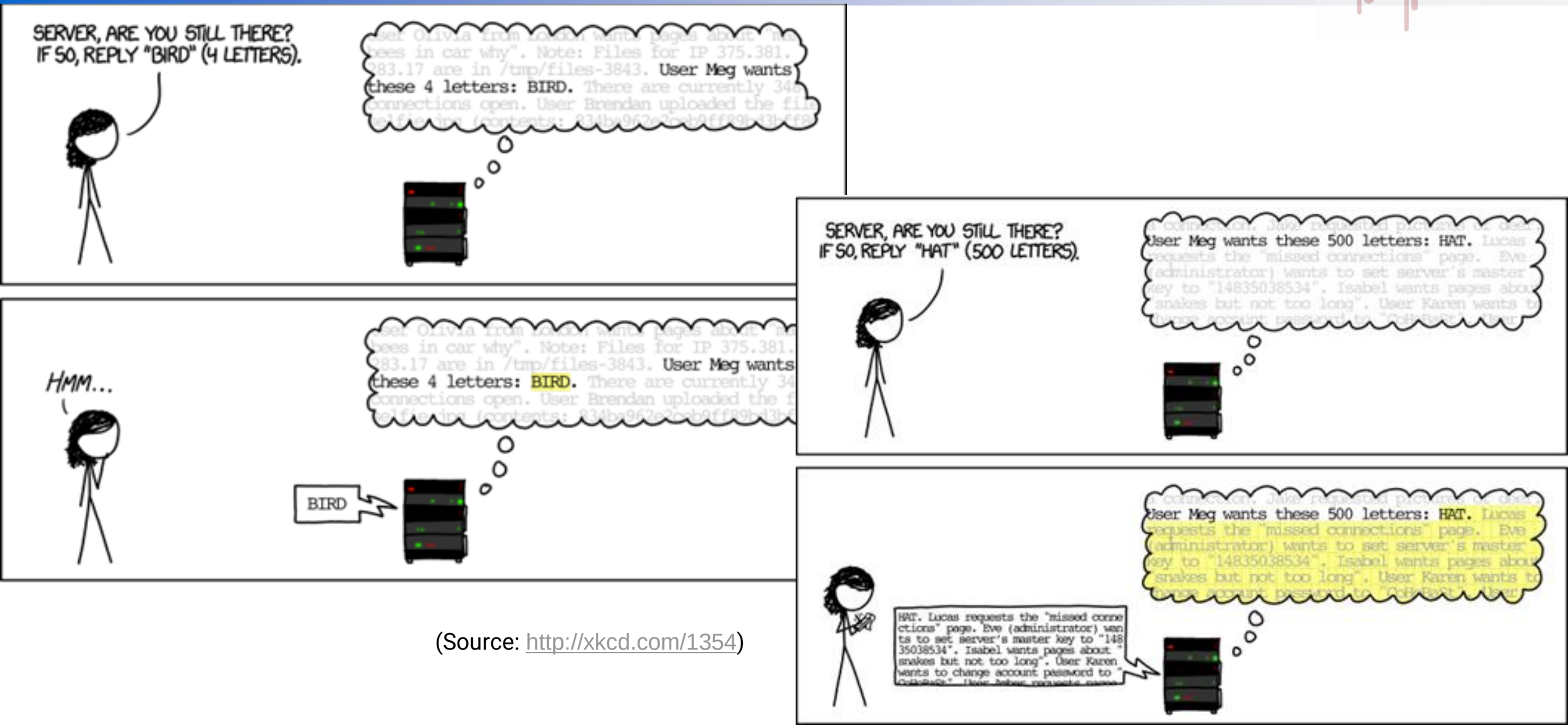
BOF: Causes, Attributes, and Consequences



BOF: Example 1 – BF Explains Techniques

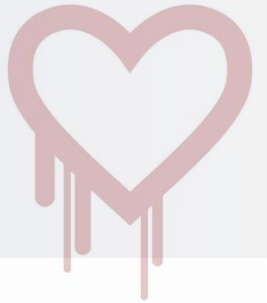
- Canaries
  - Extra memory above and below an array with unusual values, e.g., 0xDEADBEEF.
  - Useful with attributes:
    - Write *Access*
    - Small *Magnitude.*
- Address Space Layout Randomization (ASLR)
  - Allocate arrays randomly about memory.
  - Useful with attributes:
    - Heap *Location*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

BOF: Example 2 (Heartbleed)



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

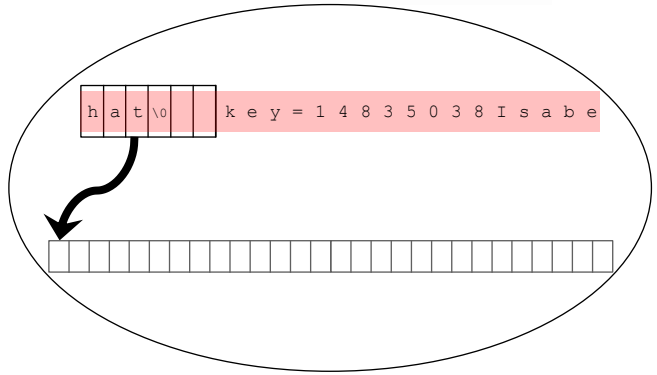
BOF: Example 2 (Heartbleed)



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.*"

[1] The MITRE Corporation, CVE Common Vulnerabilities and Exposures, CVE-2014-0160.

BOF: Example 2 – BF Description

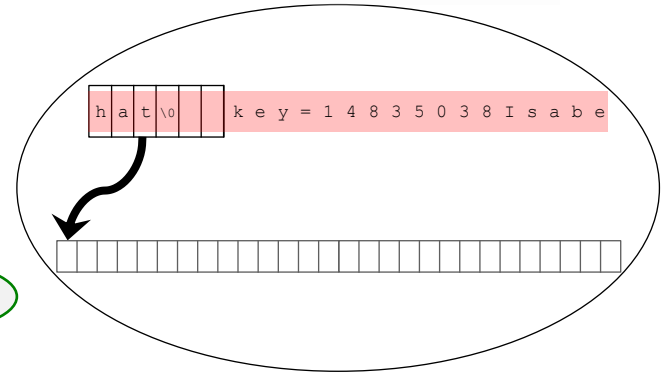


Heartbleed description using BOF taxonomy:

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

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

BOF: Example 2 (Heartbleed)



Sensitive Info Uncleared Before Release

Input Not Checked Properly

Data Exceeds Array

Array Too Small

Too Much Data

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

Information Exposure

Incorrect Calculation

Missing Factor

Integer Coercion

Integer Overflow Wrap-around

Incorrect Argument

Integer Underflow

Off By One

No NULL Termination

Wrong Index / Pointer Out of Range

Incorrect Conversion

Information Change/Loss

Altered Control Flow

Incorrect Results

Resource Exhaustion

System Crash

Program Crash

Denial Of Service

Arbitrary Code Execution

BOF: Example 2 – Analysis

The following analysis is based on information in [1,2,3,4].

- A user has to send to the software data, and a number called *payload* that is the length of that data. The software has to send the data received back to the user.
- In the Heartbleed attack, a malicious user gives *payload* a value that can be as large as 65535+1+2+16, and sends data having a number of bytes that is much less than payload, and can be as small as 1.
- The software stores that data in an array that it allocated for that purpose. The size of that array is much less than 65535+1+2+16.
- The software does not check the data and the value of *payload* in order to make sure that the number of bytes of data is equal to payload. The software therefore assumes that those numbers are equal, The software reads, using memcpy, *payload* consecutive bytes from that array, beginning at its first byte, (continuous reach) and sends them to the malicious user.
- This results in reading a large number of bytes beyond the end of the allocated array. The software did not clear the memory that it read from beyond the allocated array. Therefore, the data read and sent to the malicious user by the software includes sensitive information.

[1] The MITRE Corporation, CVE Common Vulnerabilities and Exposures, CVE-2014-0160.

[2] S. Cassidy. Diagnosis of the OpenSSL Heartbleed.

[3] WIKIPEDIA, The Free Encyclopedia, Heartbleed.

[4] The MITRE Corporation, CWE Common Weakness Enumeration, CWE-126 Buffer Over-read.

BOF: Example 2 – Source Code

Code With Bug

```
1
2
3
4  hbtype = *p++;
5  n2s(p, payload);
6
7
8  pl = p;
```

Code With Fix

```
1  /* Read type and payload length first */
2  if (1 +2 + 16 > s->s3->rrec.length)
3      return 0; /* silently discard */
4  hbtype = *p++;
5  n2s(p, payload);
6  if (1 + 2 + payload + 16 > s->s3->rrec.length)
7      return 0;  /* silently discard per RFC 6520 sec. 4 */
8  pl = p;
```

BF: BOF Exercises

Use BF to describe known software vulnerabilities or to identify gaps in existing repositories:

1) Ghost: BOF → CVE-2015-0235
2) Chrome: BOF → CVE-2010-1773
3) CWE gaps: BOF → Refactoring CWEs

Go to: goo.gl/9Ub9EL

Download: BF Exercises.pptx

BOF: Exercise 1 (Ghost)

Ghost: CVE-2015-0235

BOF: Exercise 1 (Ghost) – CVE-2015-0235

Create a BF description of CVE-2015-0235:

1. Examine the listed below CVE description, references [1,2,3], and source code excerpts with the bug and the fix.
2. Analyze the gathered information and come up with a BF description utilizing the BOF taxonomy (causes, attributes, and consequences).

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

[1] The MITRE Corporation, CVE Common Vulnerabilities and Exposures, CVE-2015-0235.

[2] Openwall, bringing security into open environment, Qualys Security Advisory CVE-2015-0235.

[3] Qualys Security Advisory CVE-2015-0235.

BOF: Exercise 1 – Source Code

Code With Bug

```
1  /* calculate size incorrectly*/
2  size_needed = (sizeof (*host_addr)+ sizeof (*h_addr_ptrs)
                  + strlen (name) + 1);
3
4  host_addr = (host_addr_t *) *buffer;
5  h_addr_ptrs = (host_addr_list_t *)((char *) host_addr
                  + sizeof (*host_addr));
6  hostname = (char *) h_addr_ptrs + sizeof (*h_addr_ptrs);
7  resbuf->h_name = strcpy (hostname, name);
```

Code With Fix

```
1  /* calculate size incorrectly*/
2  size_needed = (sizeof (*host_addr) + sizeof (*h_addr_ptrs)
                  + sizeof (*h_alias_ptr) + strlen (name) + 1);
3
4  host_addr = (host_addr_t *) *buffer;
5  h_addr_ptrs = (host_addr_list_t *)((char*) host_addr
                  + sizeof (*host_addr));
6  hostname = (char*) h_addr_ptrs + sizeof (*h_addr_ptrs);
7  resbuf->h_name = strcpy (hostname, name);
```

BOF: Exercise 1 – Analysis

The following analysis is based on information in [1,2,3].

- The number of bytes that can be overwritten is sizeof (char *), which is 4 bytes on a 32 bit machine, and 8 bytes on a 64 bit machine.
- In a calculation of the size needed to store certain data, the size of a char pointer is missing, resulting in array too small.
- Buffer over write is done by strcpy (continuous reach).
- Qualys developed an attack on the Exim mail server, exploiting this vulnerability, as proof of concept.
- This attack uses an initial buffer overwrite to enlarge the number in the size field of a portion of memory that is available for the next allocation.
- This modification enables a subsequent overwrite that enables write-anything-anywhere, which in turn enables overwriting Exim's Access Control Lists, which in turn enables arbitrary code execution.

[1] The MITRE Corporation, CVE Common Vulnerabilities and Exposures, CVE-2015-0235.

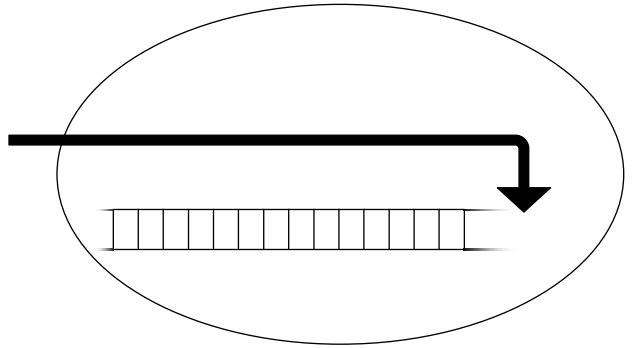
[2] Openwall, bringing security into open environment, Qualys Security Advisory CVE-2015-0235.

[3] Qualys Security Advisory CVE-2015-0235.

BOF: Exercise 1 – Solution

Ghost — gethostbyname buffer overflow is:

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



BOF: Exercise 2 (Chrome)

Chrome: CVE-2010-1773

BOF: Exercise 2 (Chrome) – CVE-2010-1773

Create a BF description of CVE-2010-1773:

1. Examine the listed below CVE description, references [1-8], and source code excerpts with bug and fix.
2. Analyze the gathered information and come up with a BF description utilizing the BOF taxonomy.

CVE-2010-1773 (Chrome WebCore): “Off-by-one error in the toAlphabetic function in rendering/RenderListMarker.cpp in WebCore in WebKit before r59950, as used in Google Chrome before 5.0.375.70, allows remote attackers to obtain sensitive information, cause a denial of service (memory corruption and application crash), or possibly execute arbitrary code via vectors related to list markers for HTML lists, aka rdar problem 8009118.” [1]

[1] The MITRE Corporation, CVE Common Vulnerabilities and Exposures, CVE-2010-1773.
[2] Robin Gandhi, Buffer Overflow Semantic template CVE-2010-1773.
[3] Tracker, Issue 44955.
[4] chromium, Diff of /branches/WebKit/375/WebCore/rendering/RenderListMarker.cpp. Revision 48099.
[5] chromium, Contents of /branches/WebKit/375/WebCore/rendering/RenderListMarker.cpp. Revision 44321.
[6] chromium, Contents of /branches/WebKit/375/WebCore/rendering/RenderListMarker.cpp. Revision 48100.
[7] webkit, Fix for Crash in WebCore::toAlphabetic() while running MangleMe -and corresponding- https://bugs.webkit.org/show_bug.cgi?id=39508. Reviewed by Darin Adler.
[8] Hat Bugzilla – Bug 596500- (CVE-2010-1773) CVE-2010-1773 WebKit: off-by-one memory read out of bounds vulnerability in handling of HTML lists.

BOF: Exercise 2 – Source Code

Code With Bug

```
1  if (type == AlphabeticSequence)
2  {
3      while ((numberShadow /= sequenceSize) > 0)
4      {
5          letters[lettersSize  - ++length] = sequence[numberShadow % sequenceSize - 1];
6      }
7  }
```

Code With Fix

```
1  if (type == AlphabeticSequence)
2  {
3      while ((numberShadow /= sequenceSize) > 0)
4      {
5          --numberShadow;
6          letters[lettersSize - ++length] = sequence[numberShadow % sequenceSize];
7      }
8  }
```

BOF: Exercise 2 – Analysis

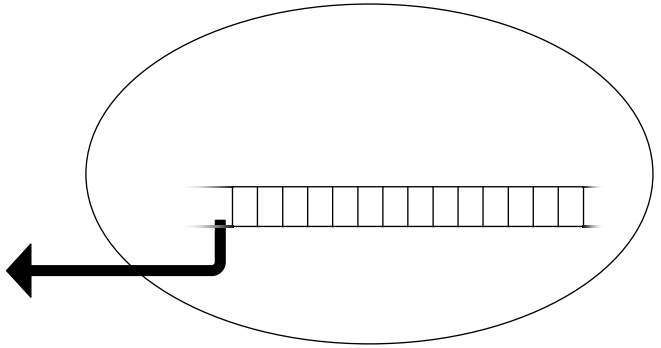
The following analysis is based on information in [1-8].

- The software reads in a loop from an array, where the sequence of indices of array elements read is neither necessarily monotonic nor necessarily having a fixed distance between consecutive elements.
- That index should be the remainder obtained by dividing an integer by an integer.
- The software subtracts 1 from that remainder, which is wrong, and can result in the index being equal to -1, leading to reading from an address that is below the beginning of the array by 1.
- Consequences are mentioned in [10], and [16] includes "An off by one memory read out of bounds issue exists in WebKit's handling of HTML lists. Visiting a maliciously crafted website may lead to an unexpected application termination or the disclosure of the contents of memory.“

[1] The MITRE Corporation, CVE Common Vulnerabilities and Exposures, CVE-2010-1773.
[2] Robin Gandhi, Buffer Overflow Semantic template CVE-2010-1773.
[3] Tracker, Issue 44955.
[4] chromium, Diff of /branches/WebKit/375/WebCore/rendering/RenderListMarker.cpp. Revision 48099.
[5] chromium, Contents of /branches/WebKit/375/WebCore/rendering/RenderListMarker.cpp. Revision 44321.
[6] chromium, Contents of /branches/WebKit/375/WebCore/rendering/RenderListMarker.cpp. Revision 48100.
[7] webkit, Fix for Crash in WebCore::toAlphabetic() while running MangleMe -and corresponding- https://bugs.webkit.org/show_bug.cgi?id=39508. Reviewed by Darin Adler.
[8] Hat Bugzilla – Bug 596500- (CVE-2010-1773) CVE-2010-1773 WebKit: off-by-one memory read out of bounds vulnerability in handling of HTML lists.

BOF: Exercise 2 – Solution

BF Description:



Chrome WebCore — render buffer overflow is:

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

BOF: Exercise 3

CWE Gaps: Refactoring BOF CWEs

BOF: Exercise 3 (Refactoring CWEs)

CWE-120: Buffer Copy without Checking Size of Input: 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.
CWE-121: Stack-based Buffer Overflow
CWE-122: Heap-based Buffer Overflow
CWE-123: Write-what-where Condition
CWE-124: Buffer Underwrite ('Buffer Underflow')
CWE-125: Out-of-bounds Read
CWE-126: Buffer Over-read
CWE-127: Buffer Under-read
CWE-786: Access of Memory Location Before Start of Buffer
CWE-787: Out-of-bounds Write
CWE-788: Access of Memory Location After End of Buffer

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

Buffer Overflow CWEs Organized by Attribute:

	Before	After	Either End		Stack	Heap
Read	127					
Write						
Either R/W		788				

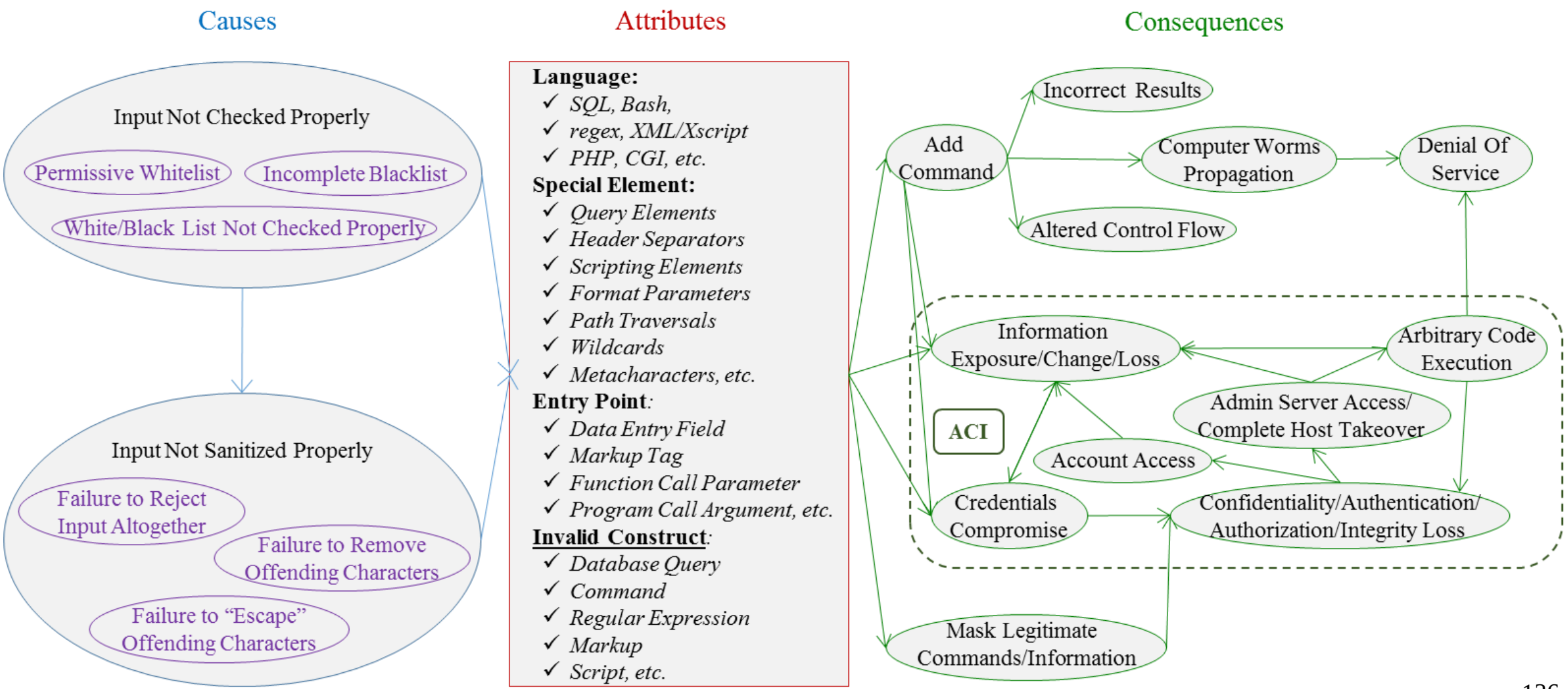
BOF: Exercise 3 – Solution

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

Buffer Overflow CWEs Organized by Attribute:

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

INJ: Causes, Attributes, and Consequences



INJ: Example

CVE 2007-3572 (Yoggie Pico): *“Incomplete blacklist vulnerability in cgi-bin/runDiagnostics.cgi in the web interface on the Yoggie Pico and Pico Pro allows remote attackers to execute arbitrary commands via shell metacharacters in the param parameter, as demonstrated by URL encoded "`" (backtick) characters (%60 sequences).” [1]*

[1] The MITRE Corporation, CVE Common Vulnerabilities and Exposures, CVE-2007-3572.

INJ: Example – CVE 2007-3572 (Yoggie Pico)

CVE 2007-3572 (Yoggie Pico) description using BOF taxonomy:

- *Input Not Checked Properly* (specifically *Incomplete Blacklist*)
- allows *Shell Metacharacters* (back ticks `)
  through a *Function Parameter* ("param") in a CGI Script
  and assembly of a string that is parsed into
  an *Invalid Command Construct* (command within a Ping command),
- which may be exploited to *Add Command*, leading to arbitrary code execution
  (adding a Ping command to change the root password enables *Complete Host Takeover*.)

This is a Shell command injection.

CVE 2007-3572 (Yoggie Pico): *"Incomplete blacklist vulnerability in cgi-bin/runDiagnostics.cgi in the web interface on the Yoggie Pico and Pico Pro allows remote attackers to execute arbitrary commands via shell metacharacters in the param parameter, as demonstrated by URL encoded "`" (backtick) characters (%60 sequences)."*

INJ: Example – Analysis

The following analysis is based on information in [1,2,3].

- Injecting backticks that are not sanitized enables adding a shell command in a CGI script.
- The Ping command was not expected to include a “command within a (Ping) command”, but the backticks (special elements) result in that unexpected structure.
- Complete Host takeover is possible by using backtick to execute changing the file /etc/shadow to include an arbitrary password selected by the attacker. Then the attacker can use that password to login as root.

[1] The MITRE Corporation, CVE Common Vulnerabilities and Exposures, CVE-2007-3572.

[2] The MITRE Corporation, CWE Common Weakness Enumeration, CWE-78: Improper Neutralization of Special Elements used in an OS Command ('OS Command Injection').

[3] Yoggie Pico Pro Remote Code Execution.

BF: INJ: Exercise – CVE-2008-5817

Use BF to describe known software vulnerabilities:

INJ → CVE-2008-5817

INJ: Exercise – CVE-2008-5817

Create a BF description of CVE-2008-5817:

1. Examine the listed below CVE description, references [1,2,3,4].
2. Analyze the gathered information and come up with a BF description utilizing the INJ taxonomy (causes, attributes, and consequences).

CVE-2008-5817: "Multiple SQL injection vulnerabilities in index.php in Web Scribble Solutions webClassifieds 2005 allow remote attackers to execute arbitrary SQL commands via the (1) user and (2) password fields in a sign_in action." [1]

[1] The MITRE Corporation, CVE Common Vulnerabilities and Exposures, CVE-2008-5817.

[2] CXSESECURITY, webClassifieds 2005 (Auth Bypass) SQL Injection Vulnerability CWE-89 CVE-2008-5817.

[3] The MITRE Corporation, CWE Common Weakness Enumeration, CWE-89: Improper Neutralization of Special Elements used in an SQL Command ('SQL Injection').

[4] Bricks, SQL injection.

INJ: Exercise – Analysis

The following analysis is based on information in [1,2,3,4].

- According to [3], ‘ or ‘ 1=1 is used to mask password checking and login as admin.
  [4] includes an explanation of this type of SQL injection.

[1] The MITRE Corporation, CVE Common Vulnerabilities and Exposures, CVE-2008-5817.

[2] CXSECURITY, webClassifieds 2005 (Auth Bypass) SQL Inje ction Vulnerability CWE-89 CVE-2008-5817.

[3] The MITRE Corporation, CWE Common Weakness Enumeration, CWE-89: Improper Neutralization of Special Elements used in an SQL Command ('SQL Injection').

[4] Bricks, SQL injection.

INJ: Exercise – Solution

CVE-2008-5817 description using BOF taxonomy:

- *Input Not Checked Properly* or *Input Not Sanitized Properly*
- allows *SQL Query Elements* (specifically single quote ‘ , the word or, and equality sign =) through *Data Entry Fields* (“username“ & “password“) in a PHP script and assembly of a string that is parsed into an *Invalid Database Query Construct*,
- which may be exploited to *Mask Legitimate* SQL *Commands*, leading to *Authentication Compromise*, *Admin Server Access* and *Arbitrary Code Execution*.

This is SQL injection.

Session III – END

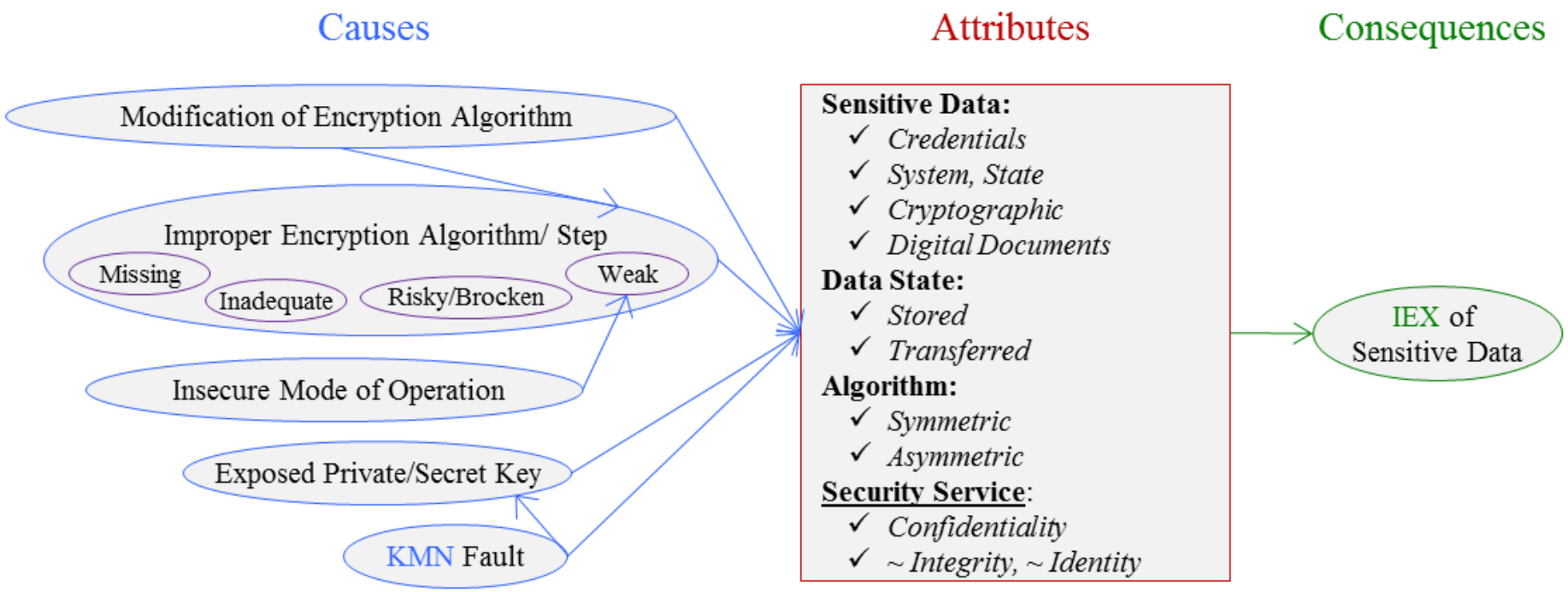
15 min BREAK

Session IV starts at 4:15pm

Session IV. “Hands-On” with Cryptography Bugs

- Encryption Bugs (ENC) Class
  - ✓ Example of Applying ENC
  - ✓ Exercise on Applying ENC
- Verification Bugs (VRF) Class
  - ✓ Example on Applying VRF
  - ✓ Exercise on Applying VRF
- Key Management Bugs (KMN) Class
  - ✓ Example on Applying KMN
  - ✓ Exercise on Applying ENC, VFR, and KMN

ENC: Causes, Attributes, and Consequences



ENC: Example

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

CVE-2002-1946 description using ENC taxonomy:

- Use of *Weak Symmetric Encryption Algorithm* (one-to-one mapping)
- allows *Confidentiality* failure
  of *Stored* (in registry) *Sensitive Data* (passwords),
- which may be exploited for *IEX of* that *Sensitive Data*.

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.“

ENC: Example – Analysis

The following analysis is based on information in [1,2,3,4].

- The one-to-one mapping uses two fixed arrays of characters. There was no remedy as of 09/01/2014!

[1] The MITRE Corporation, CVE-2002-1946, http://cve.mitre.org/cgi-bin/cvename.cgi?name=CVE-2002-1946.

[2] The MITRE Corporation, CWE 326, https://cwe.mitre.org/data/definitions/326.html.

[3] SecurityTracker. VSNL Integrated Dialer Weak Encoding Discloses Passwords to Local Users Alert ID: 1005515, http://securitytracker.com/id/1005515.

[4] IBM X-Force Exchange, Integrated Dialer Software stores passwords using weak encryption algorithm: CVE-2002-1946, https://exchange.xforce.ibmcloud.com/vulnerabilities/10517.

BF: ENC Exercise

Use BF to describe known software vulnerabilities:

ENC → CVE-2002-1697

ENC: Exercise – CVE-2002-1697

Create a BF description of CVE-2002-1697:

1. Examine the listed below CVE description, as well as references [1,2,3,4].
2. Analyze the gathered information and come up with a BF description utilizing the ENC taxonomy (causes, attributes, and consequences).

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.
[2] Wikipedia, RSA (cryptosystem), http://en.wikipedia.org/wiki/RSA_(cryptosystem).
[3] Seclists, Security weaknesses of VTun, http://seclists.org/bugtraq/2002/Jan/119.
[4] Wikipedia, Deterministic encryption, https://en.wikipedia.org/wiki/Deterministic_encryption.

ENC: Exercise – Analysis

The following analysis is based on information in [1,2,3,4].

- Using electronic codebook (ECB) results in weak encryption, that produces the same ciphertext from the same plaintext blocks. This is a case of deterministic encryption, where patterns in plaintext become evident in the ciphertext.

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

[2] Wikipedia, RSA (cryptosystem), http://en.wikipedia.org/wiki/RSA_(cryptosystem).

[3] Seclists, Security weaknesses of VTun, http://seclists.org/bugtraq/2002/Jan/119.

[4] Wikipedia, Deterministic encryption, https://en.wikipedia.org/wiki/Deterministic_encryption.

ENC: Exercise – Solution

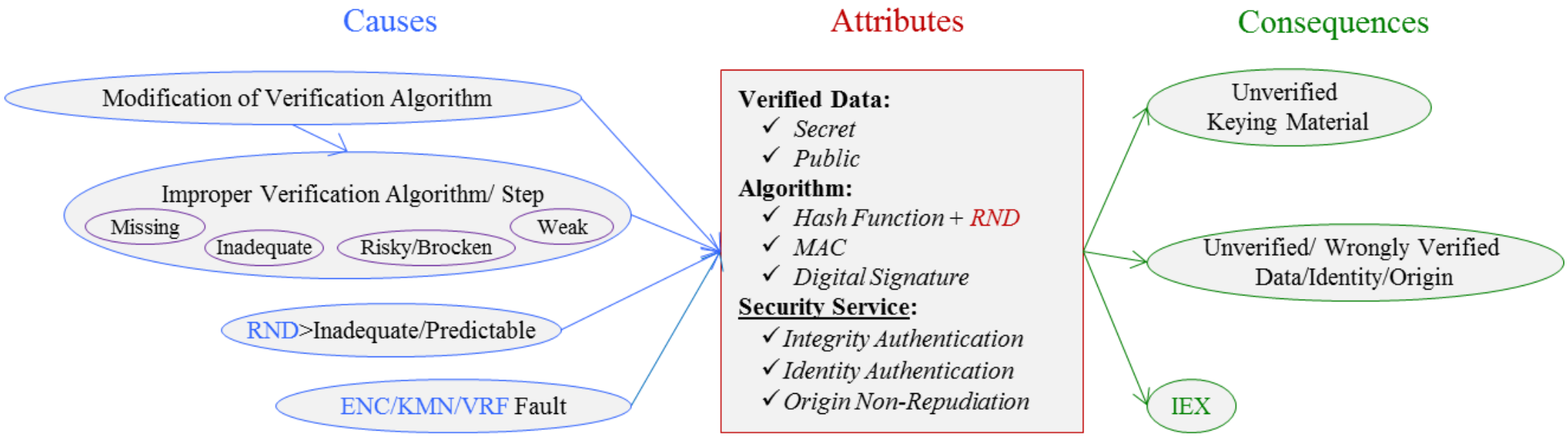
CVE-2002-1697 description using ENC taxonomy :

- Use of *Insecure Mode of Operation* (ECB) leads to *Weak Symmetric Encryption Algorithm* (for same shared key produces same ciphertext from same plaintext)
- that allows *Confidentiality* failure of *Transferred Sensitive Data*,
- which may be exploited for *IEX* of that *Sensitive Data*.

→ **Note from an attendee:**

→ **ALSO ~ Integrity due to the ECB mode – IMPORTANT!!!**

VRF: Causes, Attributes, and Consequences



VRF: Example – 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 – CVE-2001-1585

CVE-2001-1585 description using VRF taxonomy:

- *Missing Verification Step* in public key authentication (challenge-response of private key using *Digital Signature*)
- allows *Identity Authentication* failure,
- which 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.“

VRF: Example – Analysis

The following analysis is based on information in [1,2,3].

- The step that should be included is challenge-response authentication:
  - The client is required by the server to sign a message using the client's private key.
  - Successful verification of that signature by the server, using the public key, confirms that the client owns the private key that is paired with that public key, and therefore that client should be allowed to login.

That challenge-response authentication step is missing.

[1] The MITRE Corporation, CVE-2001-1585, http://cve.mitre.org/cgi-bin/cvename.cgi?name=CVE-2001-1585.

[2] OpenBSD Security Advisory, Authentication By-Pass Vulnerability in OpenSSH-2.3.1, http://www.openbsd.org/advisories/ssh_bypass.txt.

[3] S. Tatham, PuTTY User Manual – Chapter 8: "Using public keys for SSH authentication," http://the.earth.li/~sgtatham/putty/0.60/htmldoc/Chapter8.html.

BF: VRF Exercise – CVE-2015-2141

Use BF to describe known software vulnerabilities:

VRF → CVE-2015-2141

VRF: Exercise – CVE-CVE-2015-2141

Create a BF description of CVE-CVE-2015-2141:

1. Examine the listed below CVE description, as well as references [1,2,3,4].
2. Analyze the gathered information and come up with a BF description utilizing the ENC taxonomy (causes, attributes, and consequences).

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

[2] Bugzilla – Bug 936435, VUL-0: CVE-2015-2141: libcryptopp: libcrypto++ -- security update, https://bugzilla.suse.com/show_bug.cgi?id=936435.

[3] E. Sidorov, "Breaking the Rabin-Williams digital signature system implementation in the Crypto++ library," 2015, http://eprint.iacr.org/2015/368.pdf.

[4] Wikipedia, "Blinding Cryptography," https://en.wikipedia.org/wiki/Blinding_(cryptography).

VRF: Exercise – Analysis

The following analysis is based on information in [1,2,3,4].

- Having the private key allows an attacker to be authenticated as the owner of that key.
- The software intends to use blinding to defend against a timing attack, as follows: Instead of signing the data directly, the data is first transformed using a secret random value (blinding) and then is digitally signed using a private key. At the end, the effect is removed (unblinding), so that there is signed data as if no transformation took place. See [20, 21] for blinding used for RSA.
- The flaw in this CVE is in doing blinding/ unblinding incorrectly, so that in some cases the effect of the transformation is not removed from the data. This enables the attacker to use the transformed data to recover the private key using a mathematical calculation as described in [20]. In [20] it is observed that if the secret random integer used to transform the message is a quadratic residue modulo an appropriate integer, then the unblinding step correctly undoes the transformation. The fix in [20] assures that the integer is such a quadratic residue.

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

[2] Bugzilla – Bug 936435, VUL-0: CVE-2015-2141: libcryptopp: libcrypto++ -- security update, https://bugzilla.suse.com/show_bug.cgi?id=936435.

[3] E. Sidorov, “Breaking the Rabin-Williams digital signature system implementation in the Crypto++ library,” 2015, http://eprint.iacr.org/2015/368.pdf.

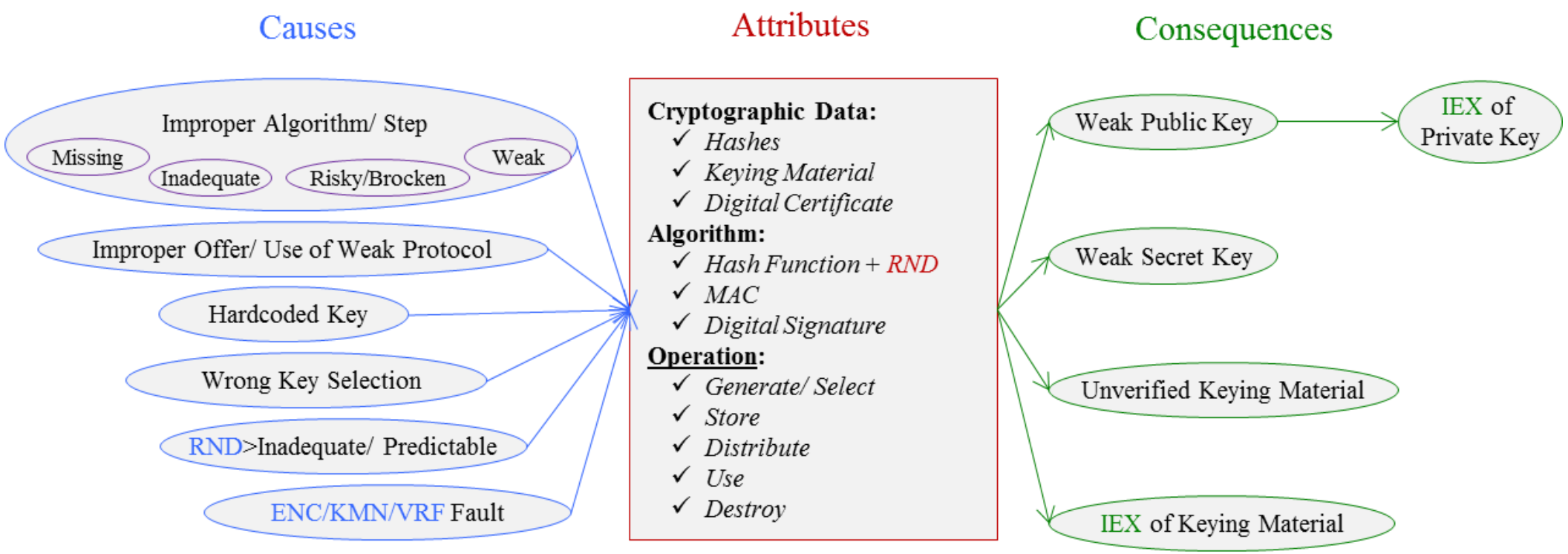
[4] Wikipedia, “Blinding Cryptography,” https://en.wikipedia.org/wiki/Blinding_(cryptography).

VRF : Exercise – Solution

CVE-2015-2141 description using VRF taxonomy:

- Modification of Digital Signature Verification Algorithm (Rabin-Williams) by adding a step (blinding) leads to recovery of private key,
- that allows *Identity Authentication* failure,
- which may be exploited for *IEX*.

KMN: Causes, Attributes, and Consequences



KMN: Example – CVE-2016-1919

CVE-2016-1919: “Samsung KNOX 1.0 uses a weak eCryptFS Key generation algorithm, which makes it easier for local users to obtain sensitive information by leveraging knowledge of the TIMA key and a brute-force attack.“ [1]

[1] The MITRE Corporation, CVE Common Vulnerabilities and Exposures, CVE-2016-1919.

KMN: Example – CVE-2002-1946

CVE-2002-1946 description using KMN and ENC taxonomy:

A KMN leads to an ENC.

KMN:

- Use of *Weak Algorithm* (eCryptFS-key from password and stored TIMA key)
- allows *Generation* of *Keying Material* (secret key) that can be obtained through brute force attack,
- which may be exploited for *IEX of* that *Keying Material* (secret key).

ENC:

- *KMN Fault* leads to *Exposed Secret Key*
- that allows *Confidentiality* failure of *Stored Sensitive Data*,
- which may be exploited for *IEX of* that *Sensitive Data*.

CVE-2016-1919:“Samsung KNOX 1.0 uses a weak eCryptFS Key generation algorithm, which makes it easier for local users to obtain sensitive information by leveraging knowledge of the TIMA key and a brute-force attack.“

KMN: Example – Analysis

The following analysis is based on information in [1,2].

- The set of possible keys is a known small set. The TIMA key is a random stored byte string.
- The secret key used is obtained by XOR of the TIMA key and the password characters, where the minimum password length is 7.
- However, if the password length is no more than 8, a base 64 expansion results in a key that does not depend on the password.
- Even if the password The TIMA key is stored, and for a known TIMA key, the key is known, or, if the password length slightly exceeds 8, there is a small set of possible keys.
- The TIMA key can be obtained using a preliminary step.

[1] The MITRE Corporation, CVE-2016-1919, https://cve.mitre.org/cgi-bin/cvename.cgi?name=CVE-2016-1919.

[2] openwall.net, [CVE-2016-1919] “Weak eCryptFS Key generation from user password,” http://lists.openwall.net/bugtraq/2016/01/17/2.

BF: KMN Exercise (FREAK)

Use BF to describe known software vulnerabilities:

FREAK: CVE-2015-0204, CVE-2015-1637, CVE-2015-1067

(FREAK - Factoring attack on RSA-ExportKeys)

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

Create a BF description for FREAK – CVE-2015-0204, CVE-2015-1637, CVE-2015-1067:

1. Examine the listed below CVE descriptions, references [1,2,3,4,5,6,7], and source code with bug and fix.
2. Analyze the gathered information and come up with a BF description utilizing the CRY taxonomy.

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.

[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.

BF: KMN Exercise (FREAK) – Source Code

Client

```
#ifndef OPENSSL_NO_RSA
if (alg_k & SSL_kRSA) {
if ((rsa=RSA_new()) == NULL) {
SSLerr(SSL_F_SSL3_GET_KEY_EXCHANGE,ERR_R_MALLOC_FAILURE);
```

```
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) {
SSLerr(SSL_F_SSL3_GET_KEY_EXCHANGE,ERR_R_MALLOC_FAILURE);
```

Server

```
case SSL3_ST_SW_KEY_EXCH_B:
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;
if (s->s3->tmp.use_rsa_tmp
```

```
case SSL3_ST_SW_KEY_EXCH_B:
alg_k = s->s3->tmp.new_cipher->algorithm_mkey;
s->s3->tmp.use_rsa_tmp=0;
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.

BF: KMN Exercise (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,

BF: KMN Exercise (FREAK) – Solution

FREAK description using VRF taxonomy:

An inner KMN leads to an inner ENC, which leads to an outer ENC.

Inner KMN:

- Client-accepted *Improper Offer of Weak Protocol* (SSL with Export RSA) from MITM-tricked server
- allows *Generation* of *Keying Material* (512-bit RSA key-pair)
  for which private key can be obtained through factorization of corresponding public key,
- which may be exploited for *IEX of* the private key from that *Keying Material*.

Inner ENC:

- *KMN Fault* leads to *Exposed Private Key* for *Asymmetric Encryption* (RSA)
- that allows *Confidentiality* failure of *Transferred Sensitive Data* (Pre-Master Secret),
- which may be exploited for *IEX of Sensitive Data* (Master Secret).

Outer ENC:

- *ENC Fault* leads to *Exposed Secret Key* (Master Secret) for *Symmetric Encryption* (RSA)
- allows *Confidentiality* failure of *Transferred Sensitive Data* (passwords, credit cards, etc.),
- which may be exploited for *IEX of* that *Sensitive Data* .

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

BF: KMN Exercise (FREAK) – Solution

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.

Questions

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

BF: INJ: Exercise – CVE-2008-5734

Use BF to describe known software vulnerabilities:

1) INJ → CVE-2008-5734

INJ: Exercise – CVE-2008-5734

Create a BF description of CVE-2008-5734:

1. Examine the listed below CVE description, references [1,2,3].
2. Analyze the gathered information and come up with a BF description utilizing the INJ taxonomy (causes, attributes, and consequences).

CVE-2008-5734: “Cross-site scripting (XSS) vulnerability in WebMail Pro in IceWarp Software Merak Mail Server 9.3.2 allows remote attackers to inject arbitrary web script or HTML via an IMG element in an HTML e-mail message.” [1]

[1] The MITRE Corporation, CVE Common Vulnerabilities and Exposures, CVE-2008-5734.

[2] SecurityFocus, Merak Mail Server and Webmail Email Message HTML Injection Vulnerability.

[3] The MITRE Corporation, CWE Common Weakness Enumeration, CWE-79: Improper Neutralization of Input During Web Page Generation ('Cross-site Scripting').

INJ: Exercise – Analysis

The following analysis is based on information in [1,2,3].

- According to [2], one of the several consequences is unauthorized access to cookie-based authentication credentials.

[1] The MITRE Corporation, CVE Common Vulnerabilities and Exposures, CVE-2008-5734.

[2] SecurityFocus, Merak Mail Server and Webmail Email Message HTML Injection Vulnerability.

[3] The MITRE Corporation, CWE Common Weakness Enumeration, CWE-79: Improper Neutralization of Input During Web Page Generation ('Cross-site Scripting').

INJ: Exercise – Solution

CVE 2008-5734 description using BOF taxonomy:

- *Input Not Sanitized Properly*
- allows *Scripting Elements* (angular brackets < >)
  through *???*
  (where is the "user-supplied input" entered from? (see
  http://www.securityfocus.com/bid/32969/discuss)
  Where is the sanitization supposed to happen – some entry fields, function parameters?
  and assembly of a string that is parsed into
  an *Invalid Markup Construct* (IMG element with script in it)
  in generated HTML email,
- which may be exploited to *Add Commands*
  or for cookie-based authentication *Credentials Compromise*,
  leading to *Arbitrary Code Execution*.

This is XSS web script injection or HTML injection.