The Bugs Framework (BF) “Hands-On”

Software developer’s and tester’s “Best Friend”

Irena Bojanova   Paul E. Black

National Institute of Standards and Technology (NIST)

Tutorial at HoTSoS, April, 2017

SAMATE

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

Introduction

- Advancements of scientific foundation in cyber-security rely on availability of accurate, precise, and non-ambiguous:
  - ✓ definitions of software weaknesses (bugs) and
  - ✓ descriptions of software vulnerabilities.

- This tutorial will demonstrate how you can more accurately and quickly diagnose and describe vulnerabilities using the Bugs Framework (BF).

Outline

- Problems With Current Bug Descriptions
- Need for Structured Approach
- The Bugs Framework (BF)
  - Developed BF Classes
  - The Buffer Overflow (BOF) Class
  - Examples of Applying BOF
  - Exercises on Applying BOF
- Other BF Classes
  - Injection (INJ), Control of Interaction Frequency (CIF), Faulty Operation (FOP), and Cryptography (CRY)
- Next Steps

Problems With Current Bug Descriptions

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

Common Weakness Enumeration (CWE)

- A “dictionary” of every *class* of bug or flaw in software. It is the best dictionary by far.
- More than 600 distinct classes, e.g.,
  - ✓ buffer overflow
  - ✓ directory traversal
  - ✓ OS injection
  - ✓ race condition
  - ✓ cross-site scripting
  - ✓ hard-coded password
  - ✓ insecure random numbers.

http://cwe.mitre.org/

However, CWE has problems:

- Definitions are imprecise and inconsistent.
- Entrees are “coarse grained” – bundle lots of stuff, e.g. consequences and likely attacks.
- 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.

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.

CWEs – Too Detailed

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

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

Software Fault Patterns (SFP) – Improve on CWEs

- SFP cluster CWEs into categories. Each cluster is factored into formally defined attributes, with sites (“footholds”), conditions, properties, sources, sinks, etc.
- SFP is a description of an family of *computations* that are:
  - ✓ Described as patterns with an invariant core and variant parts.
  - ✓ Aligned with injury.
  - ✓ Aligned with operational views and risk through events.
  - ✓ Fully identifiable in code (discernable).
  - ✓ Aligned with CWE.
  - ✓ With formally defined characteristics.

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

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

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

Summary: The program copies an input buffer to an output buffer without verifying that the size of the input buffer is less than the size of the output buffer, leading to a buffer overflow.
Extended Description: A buffer overflow condition exists when a program attempts to put more data in a buffer than it can hold, or when a program attempts to put data in a memory area outside of the boundaries of a buffer.
Common Consequences: Buffer overflows often can be used to execute arbitrary code. Buffer overflows generally lead to crashes.

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

Semantic Templates (STs) – 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. The software faults that lead to a weakness.
  2. The resources that a weakness affects.
  3. The weakness attributes.
  4. The consequences/failures resulting from the weakness.

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

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

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

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

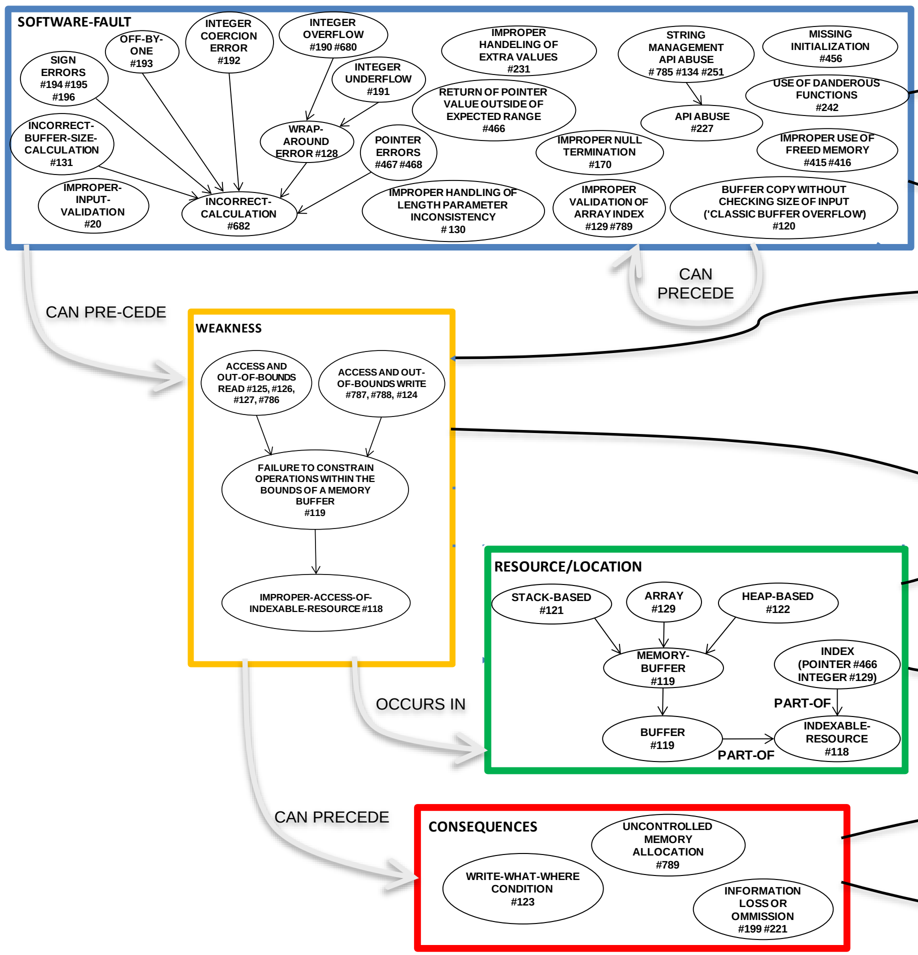
Summary: The program copies an input ***buffer*** to an output ***buffer without verifying that the size of the input buffer is less than the size of the output buffer***, leading to a ***buffer overflow***.

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

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

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

Semantic Templates (STs) – Improve on CWEs, 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 Have Problems, Too

- Software Fault Patterns (SFP):
  - ✓ "Factor" weaknesses into parameters,
  - ✓ But:
    - don't include upstream causes or consequences, and
    - are based solely on CWEs.

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

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 unnecessarily challenging.

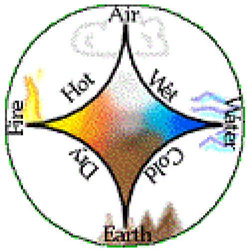
- As a result newly delivered and legacy systems will continue having security holes despite all the patching to correct errant behavior.

We are working on a good structure for bugs descriptions.

For analogies, let's look at some
well-known 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.
- Periodic table reflects atomic structure & forecasts properties of missing elements.

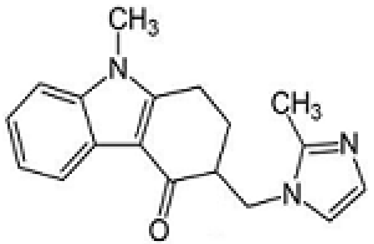


(Source: Reich Chemistry)

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



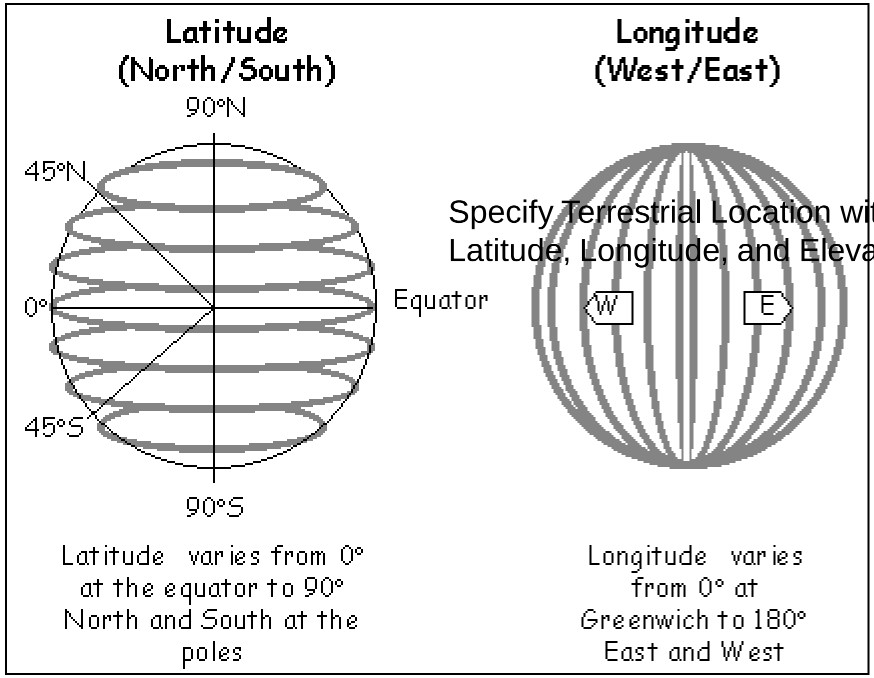
$C_{18}H_{19}N_3O$

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

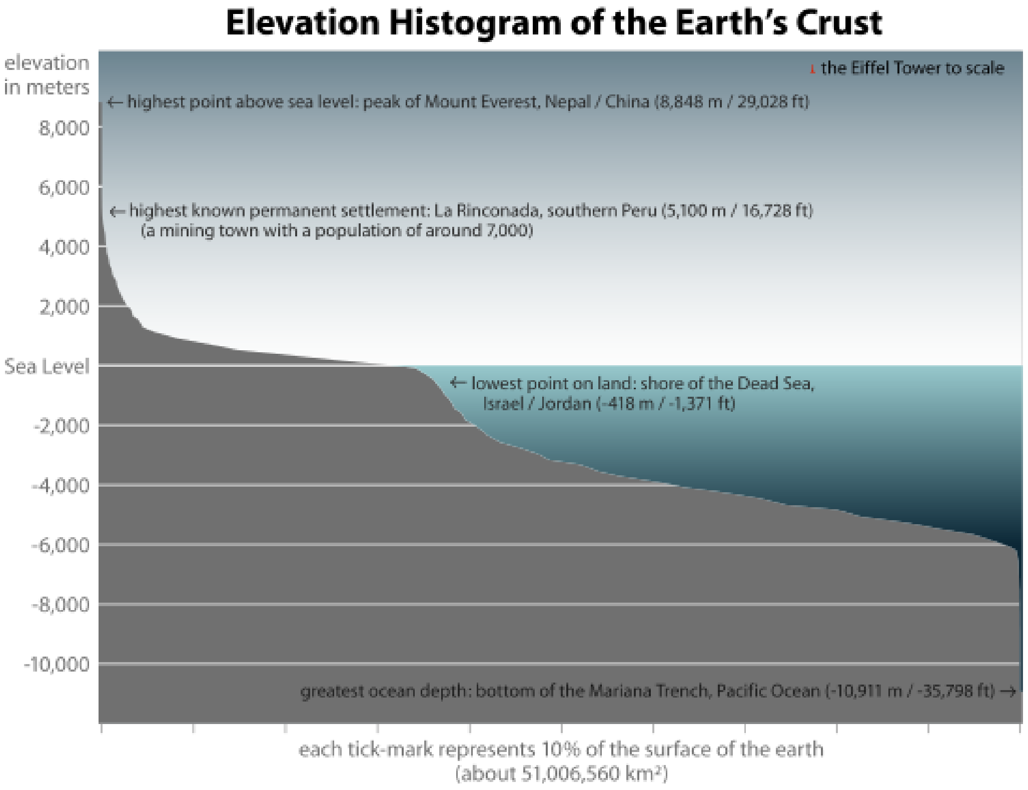
(Source: Wikimedia Commons)

Geographic Coordinate System

Specify Terrestrial Location with Latitude, Longitude, and Elevation.



Specify Terrestrial Location with Latitude, Longitude, and Elevation



Geographic Coordinate System (Source: Wikipedia)

Precise Medical Language

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

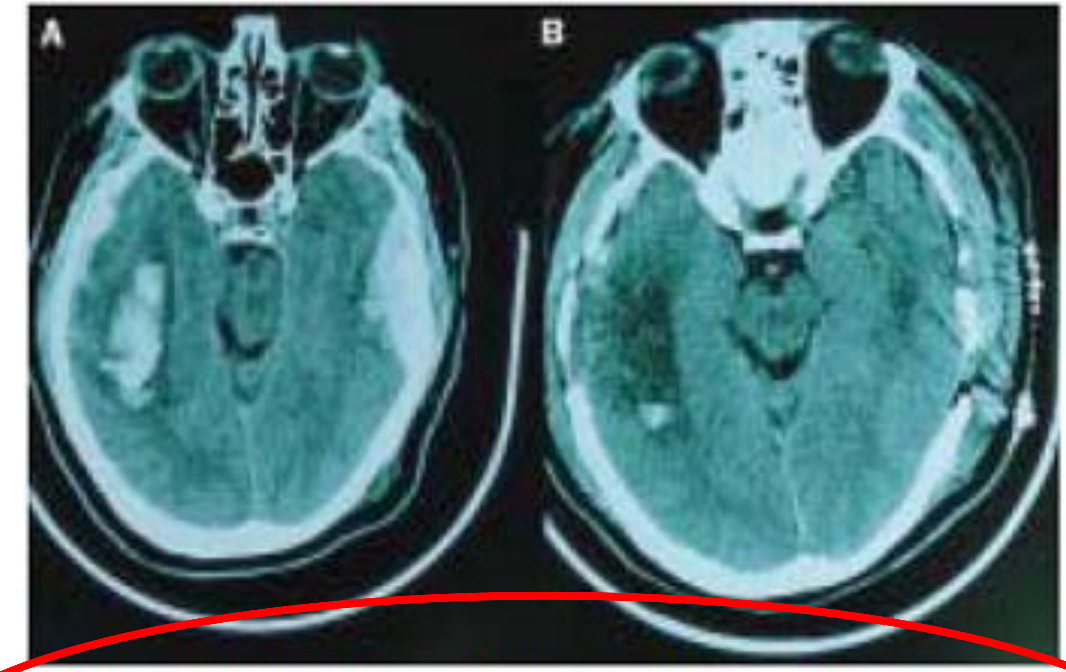


Figure 2: **Computed tomography of a comatose patient with a left temporal epidural haematoma, right parenchymal temporal lobe haematoma, and a right convexity subdural haematoma before and after craniotomy and evacuation of haematomas**

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

What is the Bugs Framework (BF)?

- It is a set of classes of faults (bad program states) caused by bugs.
- A BF bug class comprises:
  - Attributes that identify the software fault.
  - Causes that bring about the fault.
  - Consequences the fault could lead to.
  - Sites in code where the fault might occur.
- Causes and consequences as directed graphs.
- BF uses precise definitions and terminology.
- A factoring and restructuring of information in CWEs, SFPs, and STs and classifications from NSA CAS, IDA SOAR, and SEI-CERT.

What is the Bugs Framework (BF)?

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

BF Classes

- Injection (INJ), e.g.
  - ✓ SQL injection
  - ✓ OS injection.
- Control of Interaction Frequency (CIF), e.g.
  - ✓ Limit number of login attempts
  - ✓ Only one vote per voter.
- Buffer Overflow (BOF).
- Faulty Operations (FOP).
- Cryptography (CRY).
- Authentication (ATN).
- Authorization (AUT).
- Information Exposure (IEX).

BF: Buffer Overflow (BOF)

Buffer Overflow is the best class to begin.

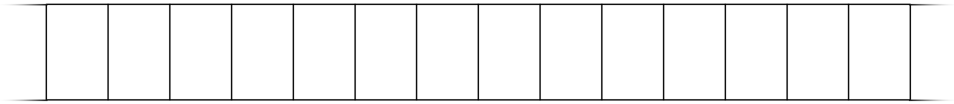
- Our Definition:
  The software accesses through an array a memory location that is outside the boundaries of that array.
  - This definition is clearer than CWE-119: Improper Restriction of Operations within the Bounds of a Memory Buffer:
    "The software performs operations on a memory buffer, but it can read from or write to a memory location that is outside of the intended boundary of the buffer."

- 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

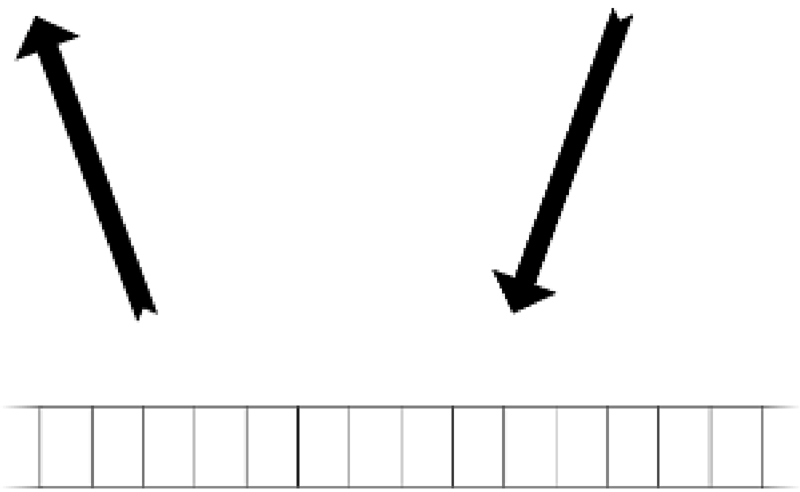
- Often referred to as a “buffer,” an array is a contiguously allocated set of objects, called elements.
  - ✓ Has a definite size – a definite number of elements are allocated to it.
  - ✓ Software should not use array name to access anything outside boundary of allocated elements.
  - ✓ Elements are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



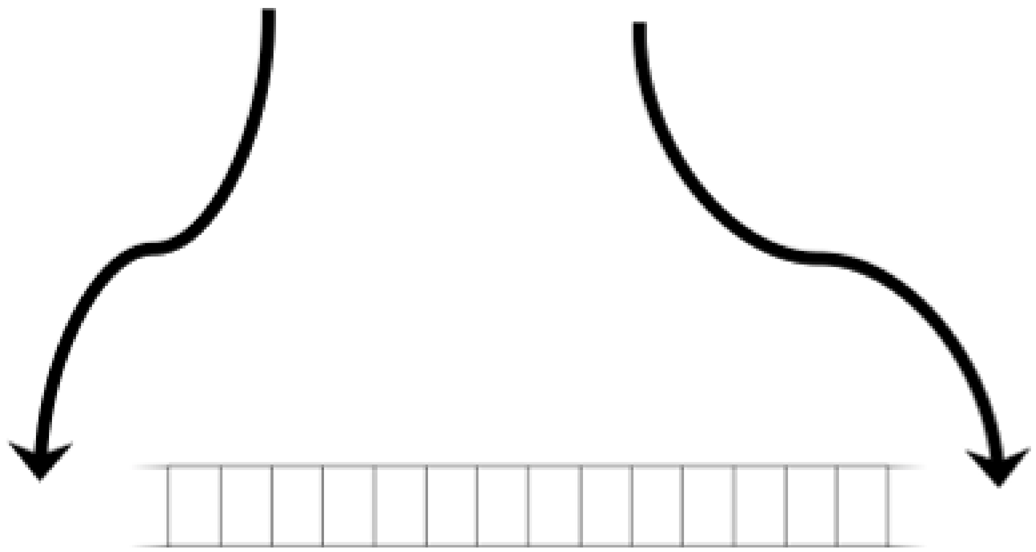
BOF Attributes – Access

- Access: Read, Write.



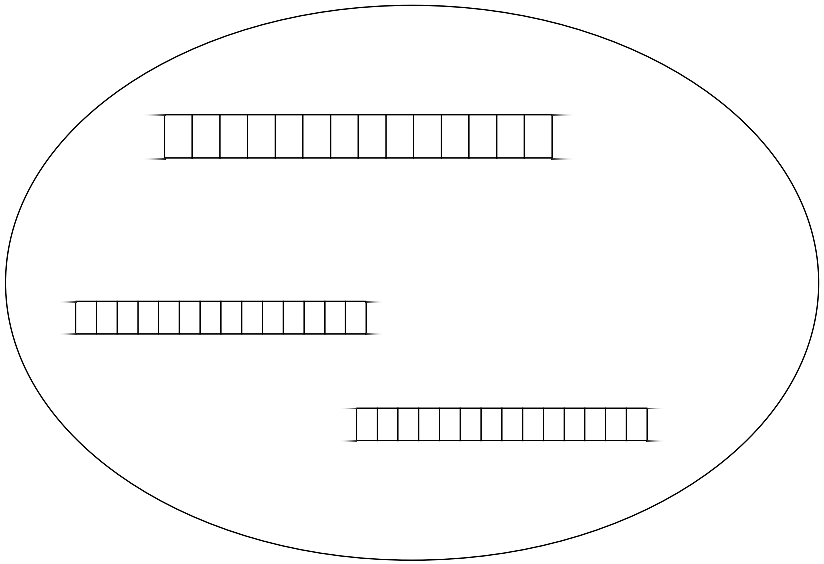
BOF Attributes – Boundary

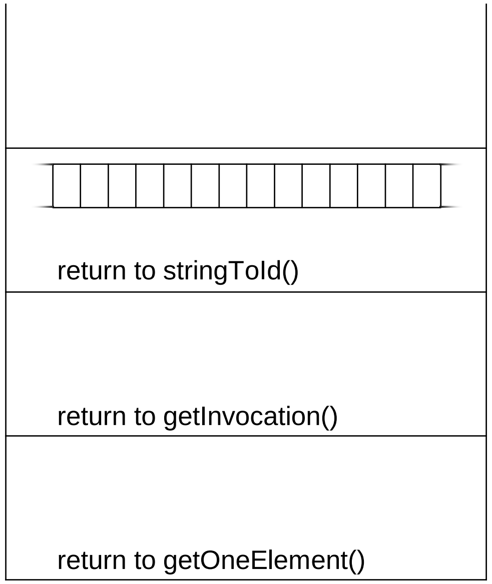
- Access: Read, Write.
- **Boundary** – which end of the array is violated:
  Below (before, under, or lower), Above (after, over, or upper).



BOF: Attributes – Location

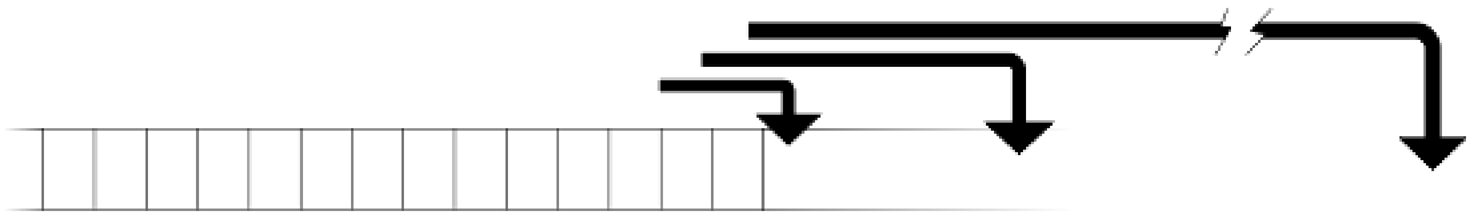
- Access: Read, Write.
- Boundary: Below, Above.
- **Location** – what part of memory the array is allocated in:
  Heap, Stack, BSS (uninitialized data), Data (initialized), Code (text), etc.





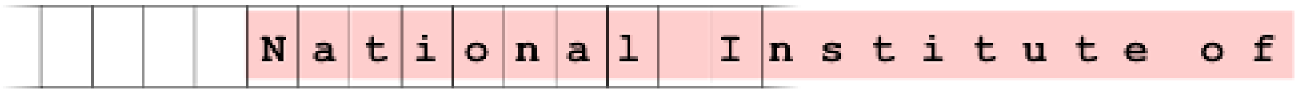
BOF Attributes – Magnitude

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



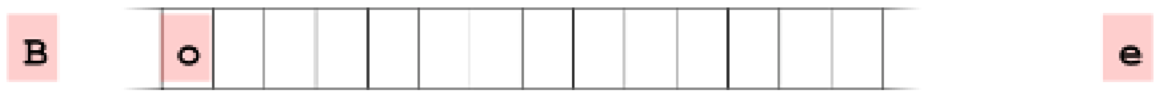
BOF Attributes – Data Size

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

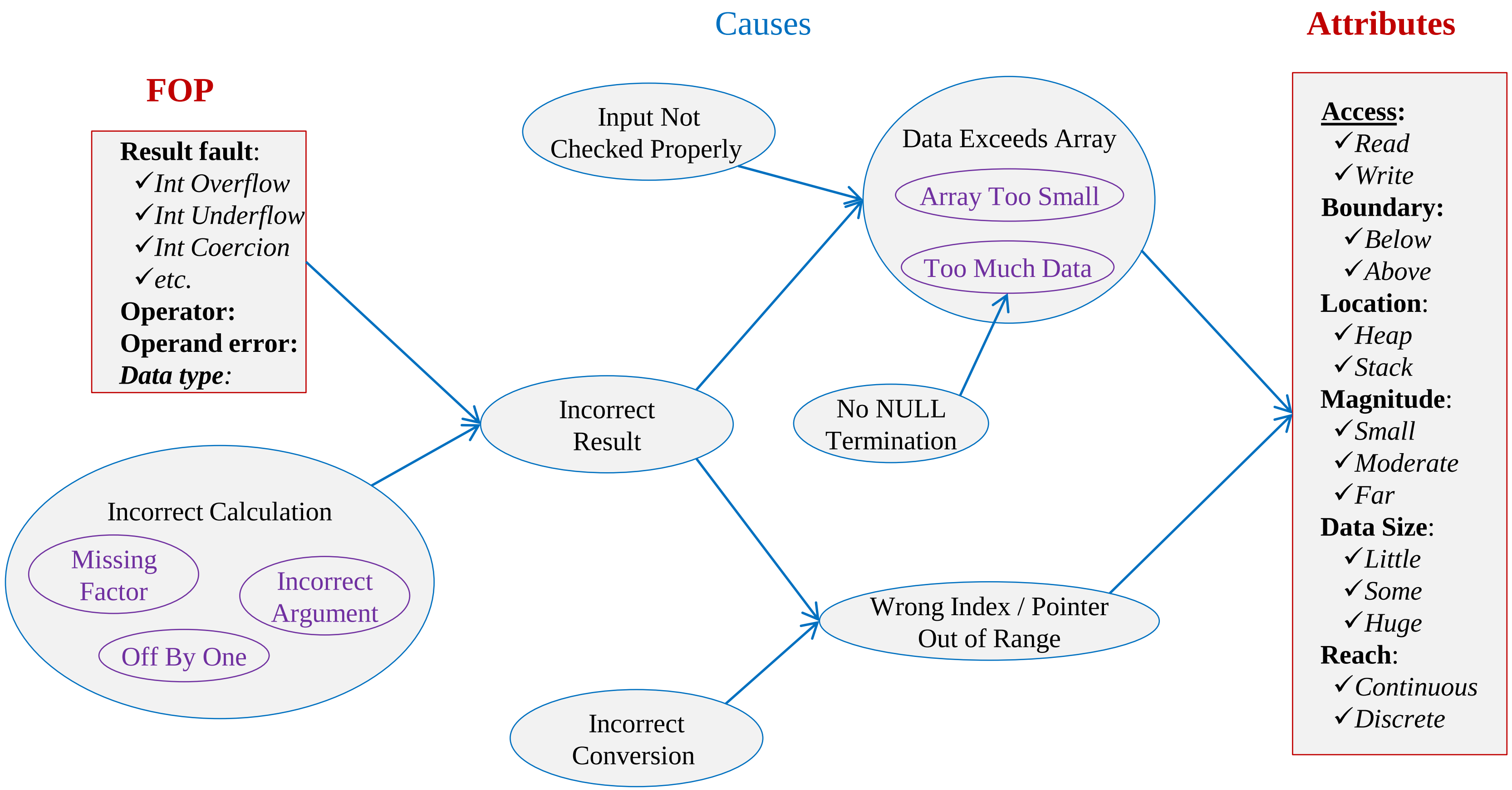


BOF Attributes – Reach

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



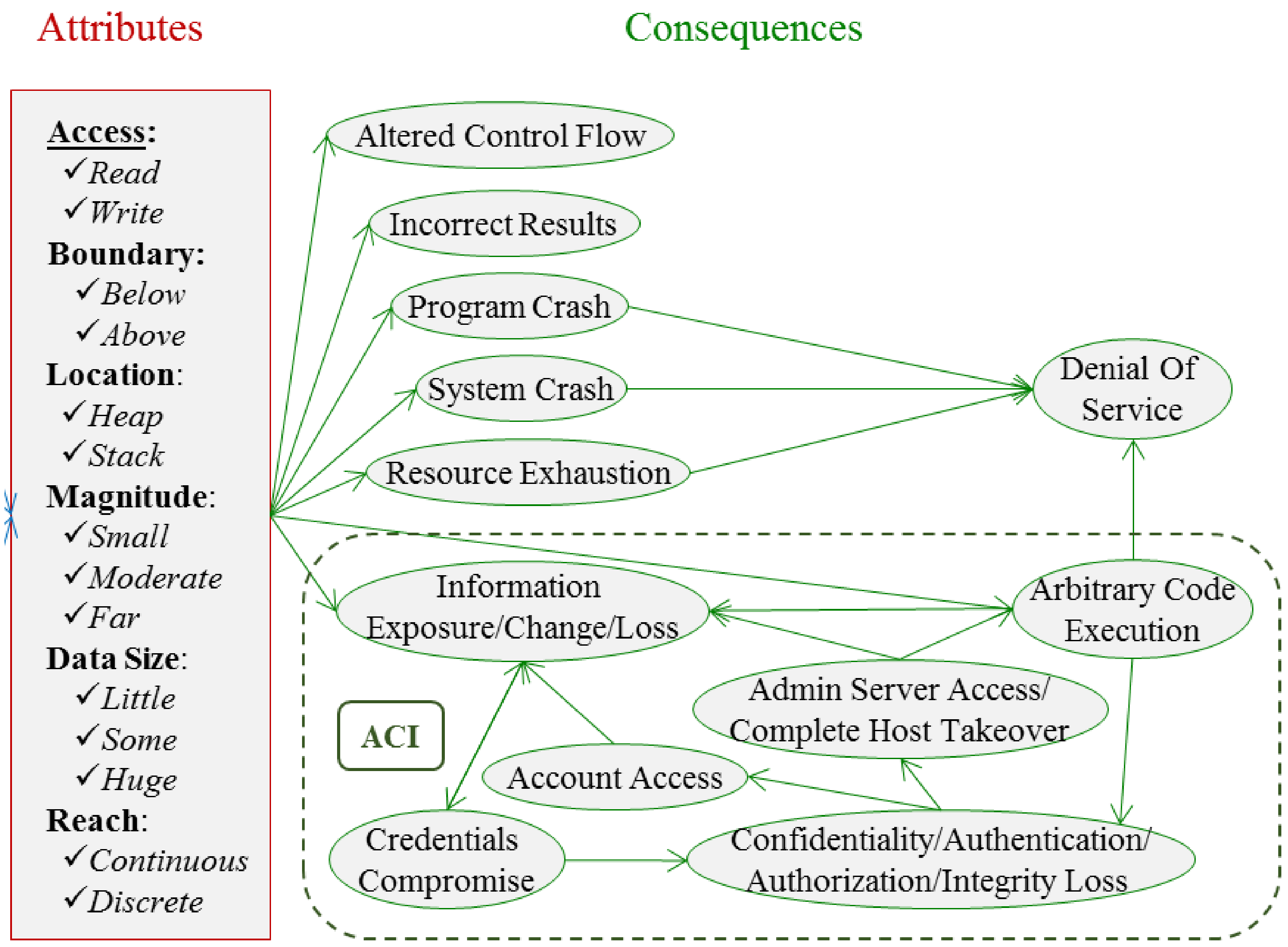
BOF: Causes



There are only two proximate causes of BOF:

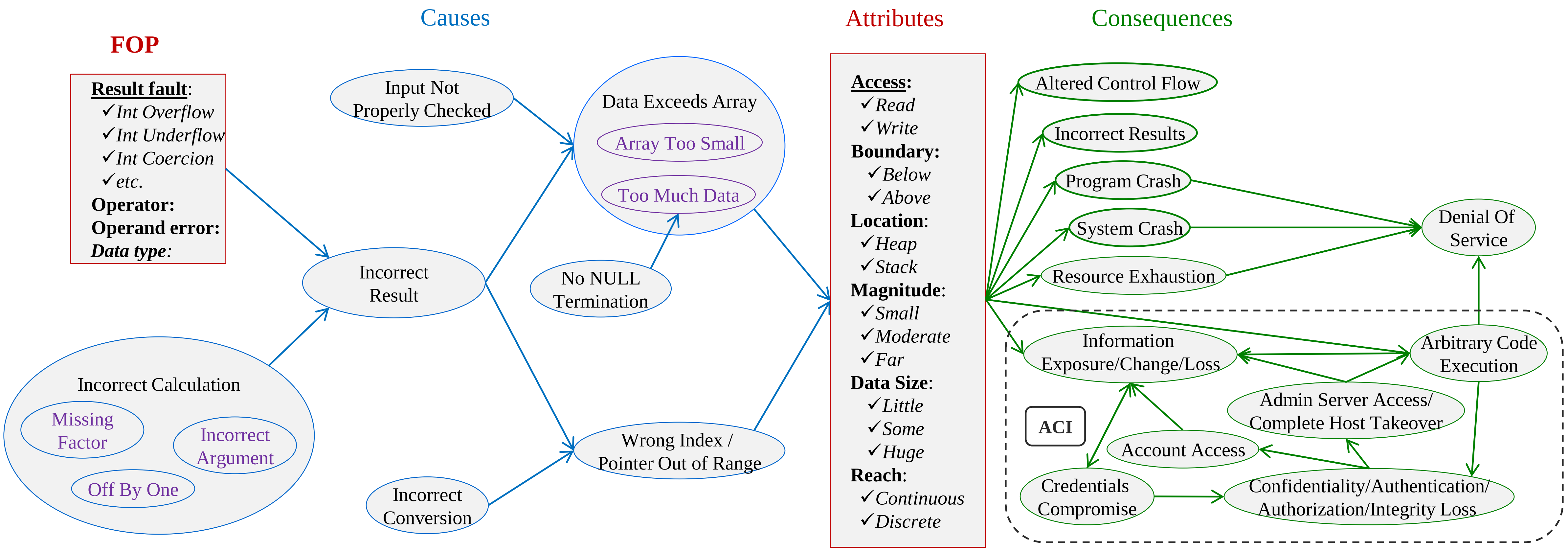
- Data Exceeds Array
- Wrong Index / Pointer Out of Range.

BOF: Consequences



Shows what could happen due to the fault.

BOF: Causes, Attributes, and Consequences



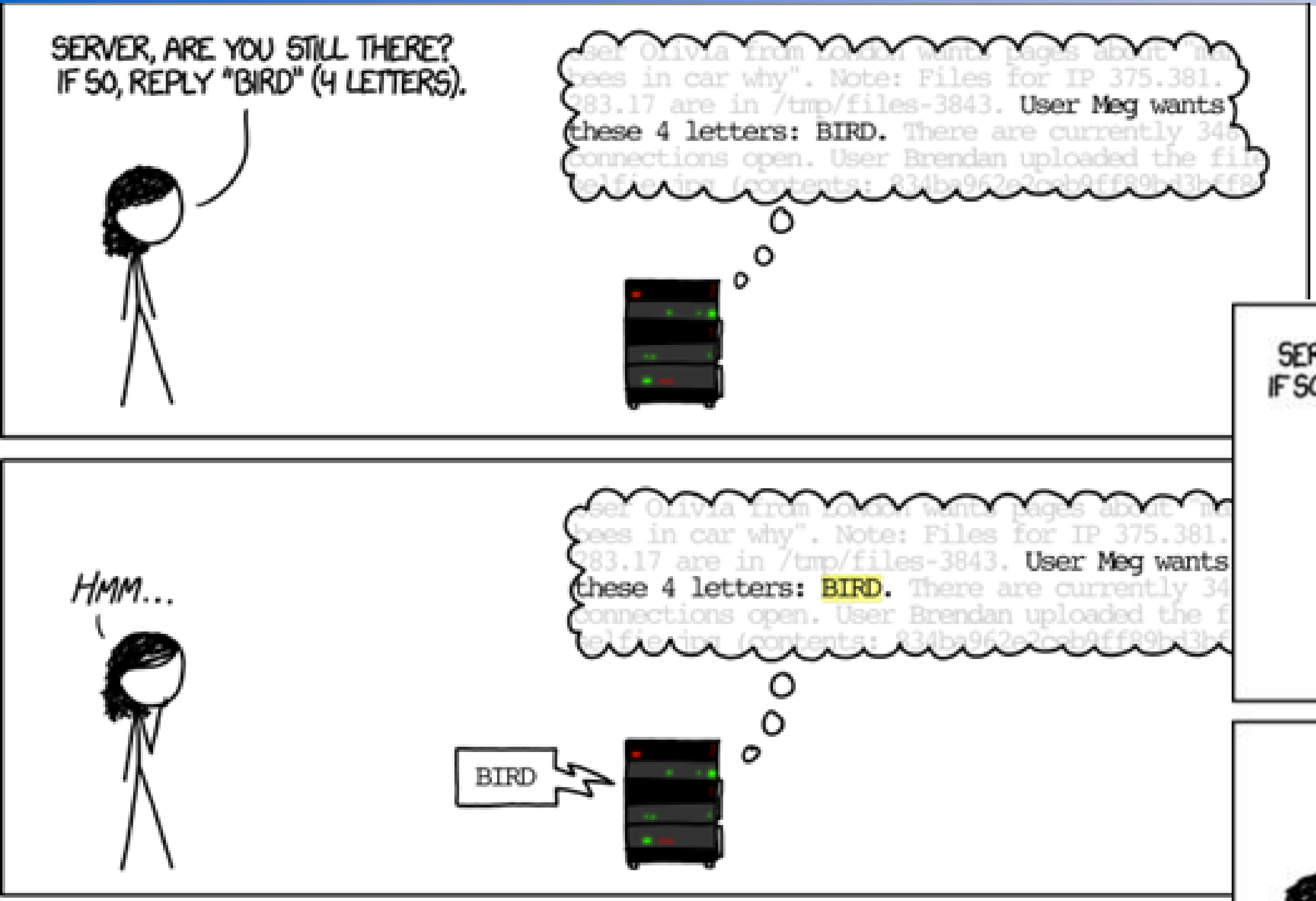
BOF: Sites

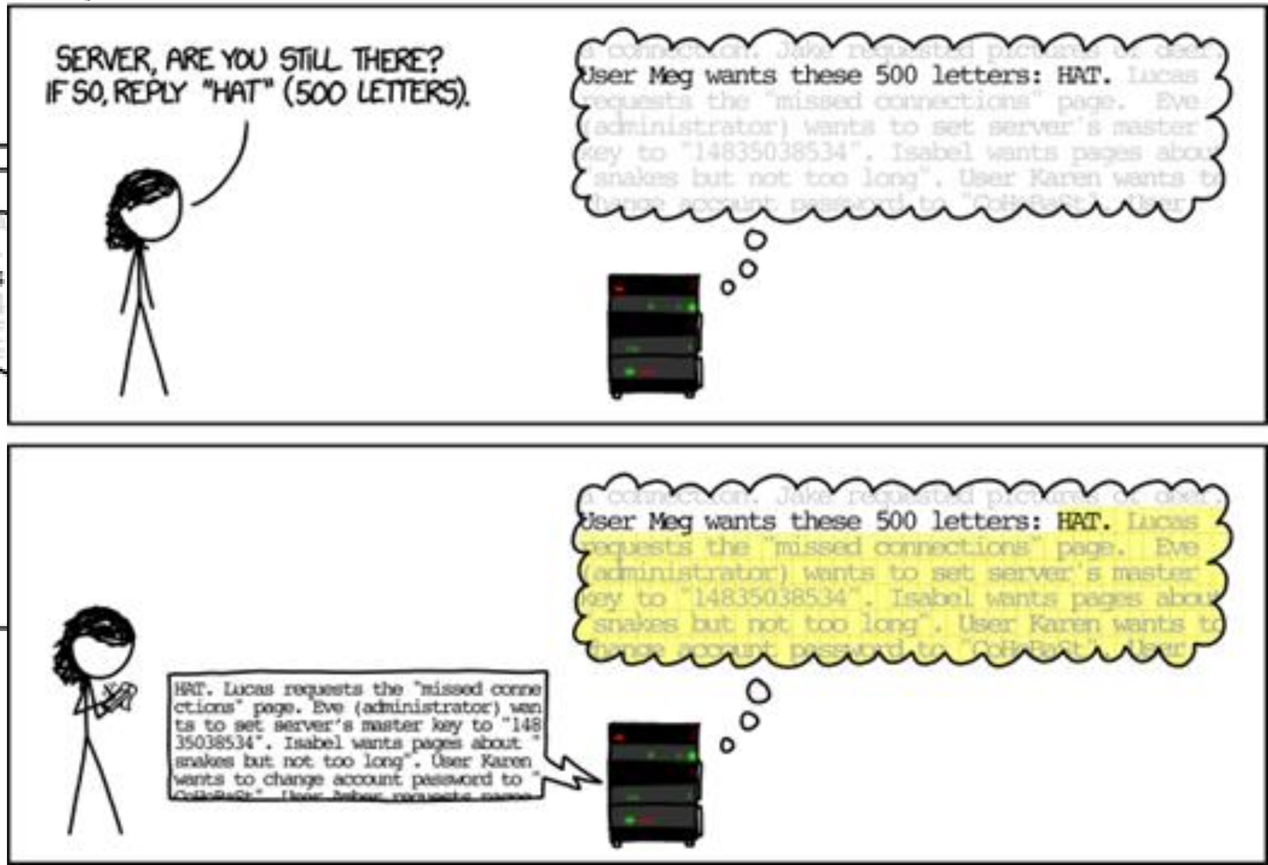
- Site: location in code that a fault may occur. *Places to check for this bug.*
- Buffer Overflow may occur at:
  - ✓ use of [ ] operator in C
  - ✓ use of unary * operator with arrays in C
  - ✓ use of string library functions, such as strcpy() or strcat().

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.
- xxxRead-only pages
- xxx(others from BOF paper)

BOF: Example 2 (Heartbleed)





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

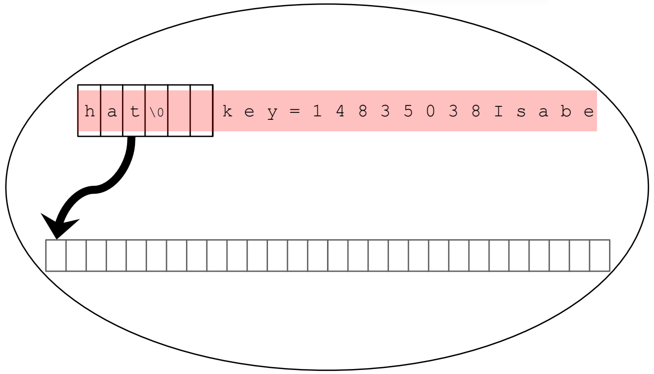
BOF: Example 2 (Heartbleed)

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

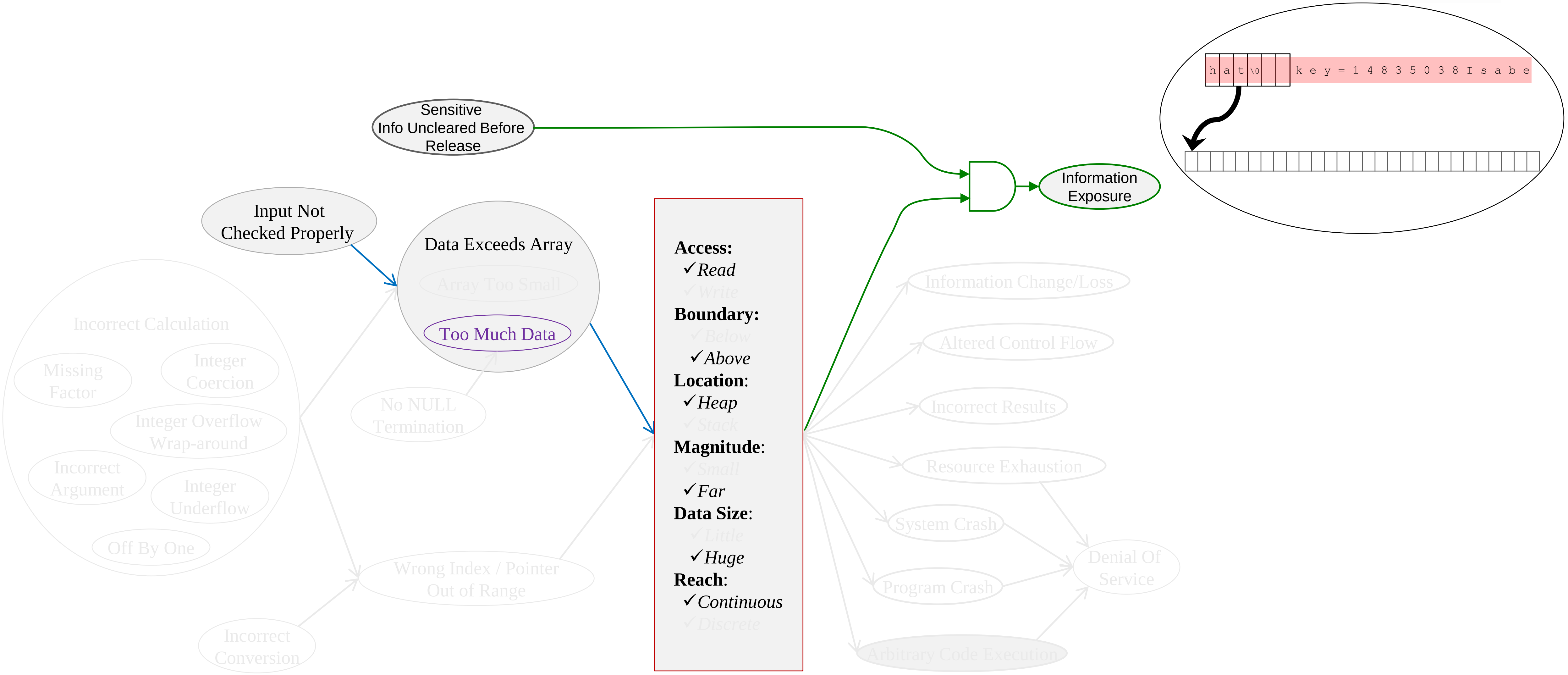
Heartbleed description using BOF taxonomy:



- *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)



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

BOF: Exercise 1 (Ghost)

Ghost: CVE-2015-0235

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

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

Create a BF description of CVE-2015-0235:

1. Examine the listed below CVE description, references, 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." [6]

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

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

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

BOF: Exercise 1 – Source Code

Code With Bug

```
/* calculate size incorrectly*/
size_needed = (sizeof (*host_addr)+ sizeof (*h_addr_ptrs)
               + strlen (name) + 1);

host_addr = (host_addr_t *) *buffer;
h_addr_ptrs = (host_addr_list_t *)((char *) host_addr
               + sizeof (*host_addr));
hostname = (char *) h_addr_ptrs + sizeof (*h_addr_ptrs);
resbuf->h_name = strcpy (hostname, name);
```

Code With Fix

```
/* calculate size incorrectly*/
size_needed = (sizeof (*host_addr) + sizeof (*h_addr_ptrs)
               + sizeof (*h_alias_ptr) + strlen (name) + 1);

host_addr = (host_addr_t *) *buffer;
h_addr_ptrs = (host_addr_list_t *)((char*) host_addr
               + sizeof (*host_addr));
hostname = (char*) h_addr_ptrs + sizeof (*h_addr_ptrs);
resbuf->h_name = strcpy (hostname, name);
```

BOF: Exercise 1 – Analysis

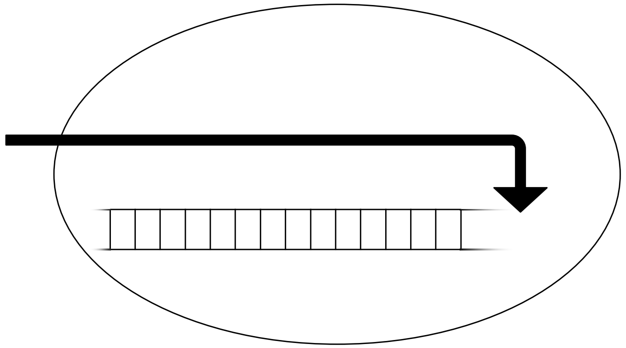
The following analysis is based on information in [6,7,8].

- 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.

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

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

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

Create a BF description of CVE-2010-1773:

1. Examine the listed below CVE description, references, 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." [9]

[9] The MITRE Corporation, CVE Common Vulnerabilities and Exposures, CVE-2010-1773.
[10] Robin Gandhi, Buffer Overflow Semantic template CVE-2010-1773.
[11] Tracker, Issue 44955.
[12] chromium, Diff of /branches/WebKit/375/WebCore/rendering/RenderListMarker.cpp. Revision 48099.
[13] chromium, Contents of /branches/WebKit/375/WebCore/rendering/RenderListMarker.cpp. Revision 44321.
[14] chromium, Contents of /branches/WebKit/375/WebCore/rendering/RenderListMarker.cpp. Revision 48100.
[15] 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.
[16] 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

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

Code With Fix

```
if (type == AlphabeticSequence)
{
    while ((numberShadow /= sequenceSize) > 0)
    {
        --numberShadow;
        letters[lettersSize - ++length] = sequence[numberShadow % sequenceSize];
    }
}
```

BOF: Exercise 2 – Analysis

The following analysis is based on information in [9-16].

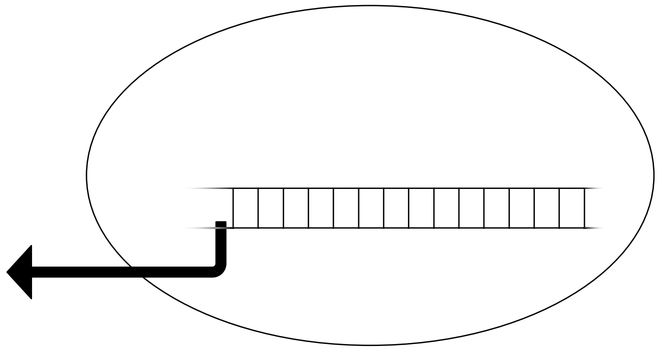
- The software reads in a loop from an array, where the sequence of indices of array elements read is neither necessarily monotonic nor necessarily having a fixed distance between consecutive elements.
- That index should be the remainder obtained by dividing an integer by an integer.
- The software 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.“

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

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

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
CWE-805: Buffer Access with Incorrect Length Value
CWE-823: Use of Out-of-range Pointer Offset

Attributes	Access		Boundary		Location		Reach	
	read	write	lower	upper	heap	stack	continuous	discrete
120		✓		✓	✓	✓	✓	
121								
122								
123								
124								
125								
126								
127								
786								
787								
788								
805								
823								

Fill in the rest from the descriptions in the handout.

BOF: Exercise 3 – Solution

CWEs refactored by BOF attribute

Attributes	Access		Boundary		Location		Reach	
	read	write	lower	upper	heap	stack	continuous	discrete
120		✓		✓	✓	✓	✓	
121		✓	✓	✓		✓		
122		✓	✓	✓	✓			
123		✓						
124		✓	✓					
125	✓		✓	✓				
126	✓			✓				
127	✓		✓					
786	✓	✓	✓					
787		✓	✓	✓				
788	✓	✓		✓				
805	✓	✓		✓			✓	
823	✓	✓	✓	✓	✓	✓		✓

BF: Injection (INJ)

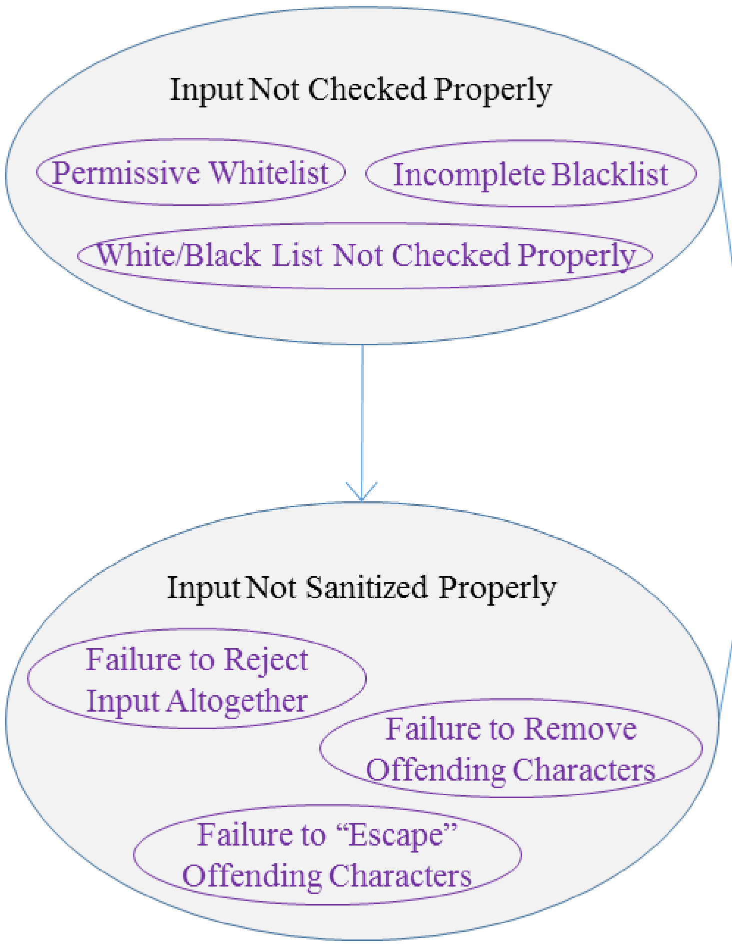
- Our Definition:
  Due to input with language-specific special elements, the software assembles a command string that is parsed into an invalid construct.

  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

Causes



Attributes

Language:

- ✓ *SQL, Bash,*
- ✓ *regex, XML/Xscript*
- ✓ *PHP, CGI, etc.*

Special Element:

- ✓ *Query Elements*
- ✓ *Header Separators*
- ✓ *Scripting Elements*
- ✓ *Format Parameters*
- ✓ *Path Traversals*
- ✓ *Wildcards*
- ✓ *Metacharacters, etc.*

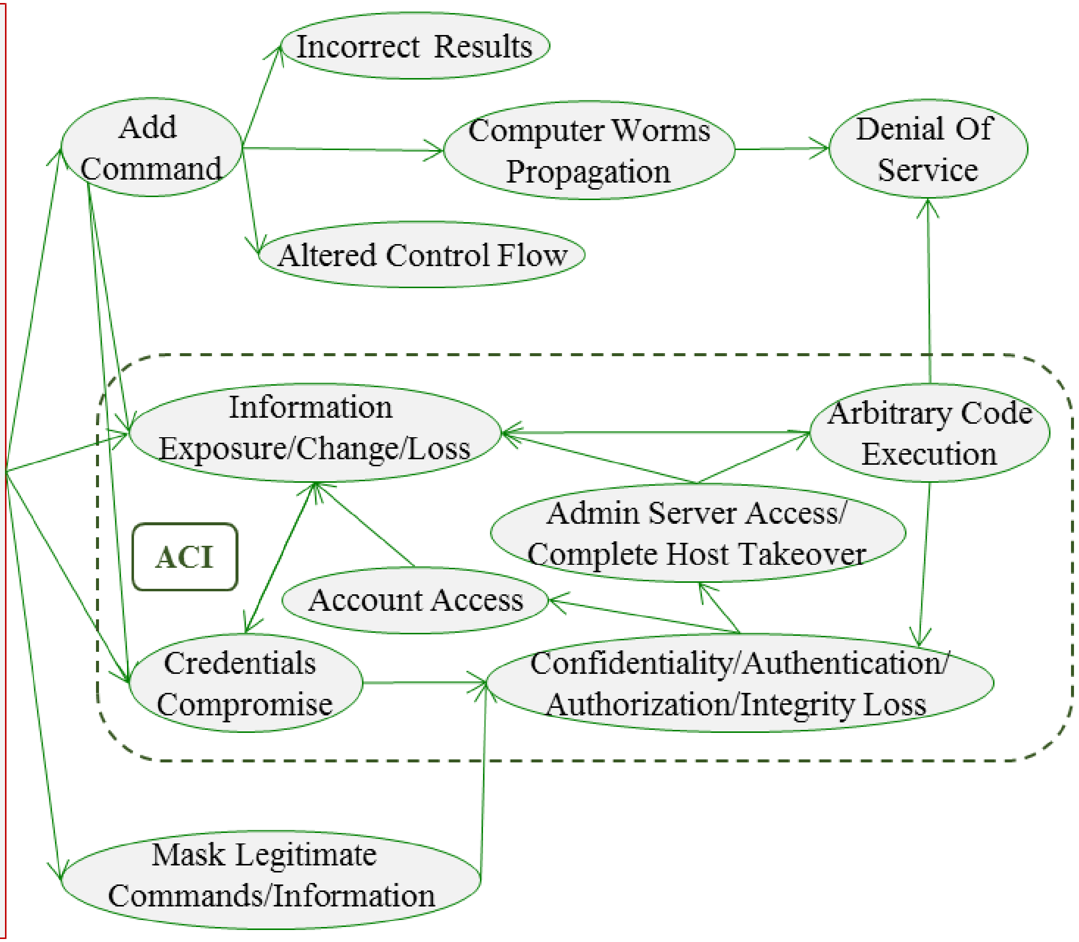
Entry Point:

- ✓ *Data Entry Field*
- ✓ *Markup Tag*
- ✓ *Function Call Parameter*
- ✓ *Program Call Argument, etc.*

Invalid Construct:

- ✓ *Database Query*
- ✓ *Command*
- ✓ *Regular Expression*
- ✓ *Markup*
- ✓ *Script, etc.*

Consequences



BF: Control of Interaction Frequency (CIF)

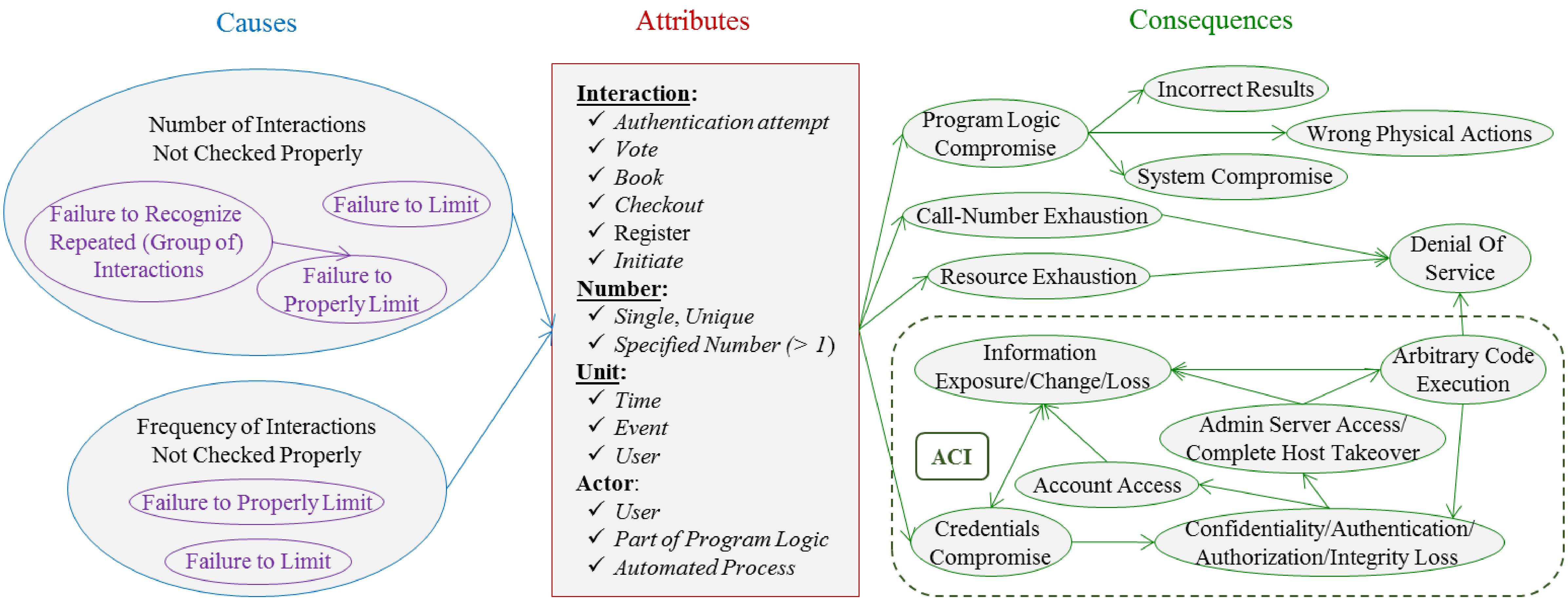
- Our Definition:
  The software does not properly limit the number of repeating interactions per specified unit.

  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



BF: Faulty Operation (FOP)

- Definition:
  Operations in the software produce an unexpected result due to range violation or conversion between primitive types.

*This is integer overflow, underflow, wrap around, divide by zero, negative shift, signed/unsigned conversion, etc. Pointer arithmetic is **not** included.*

The model is that an operation causes (implicit) conversion of its operands' values, then the operation is performed. (The C cast is an explicit conversion then a null (identity) operator. Argument passing is implicit conversion then null.)

- Related classes.
  - ✓ Related CWEs are CWE-128, CWE-190, CWE-191, CWE-192, CWE-194, CWE-195, CWE-196, CWE-197, CWE-369, and CWE-681.
  - ✓ SEI CERT Rule is Rule 04. Integers (INT), INT31-C, INT32-C, INT33-C, INT34-C, INT35-C, and INT36-C.

FOP Attributes – Result Fault

- Result Fault: value becomes small, large, undefined, etc.

  what is wrong with the result

FOP Attributes – Operator

- <u>Result Fault</u>: value becomes small, large, undefined, etc.
- Operator: arithmetic (e.g. +, /, %, --), relational (e.g. <, !=), logical (&&, ||, and !), bitwise (&, |, ~, ^, >>, and <<), conditional (?:), assignment (e.g. =, +=), explicit conversion (cast), and argument passing in a function call.

Pointer arithmetic is ***not*** *included because it mimics array accesses. Note that only additive and comparison operations (e.g. +, -, <) are defined for pointers. That is, expressions like* `pointer/int` *and* `pointer+pointer` *are not defined.*

FOP Attributes – Operand Error

- Result Fault: value becomes small, large, undefined, etc.
- Operator: arithmetic (e.g. +, /, %, --), relational (e.g. <, !=), logical (&&, ||, and !), bitwise (&, |, ~, ^, >>, and <<), conditional (?:), assignment (e.g. =, +=), explicit conversion (cast), and argument passing in a function call.
- Operand Error: mismatched data types, value too big, range error, etc.

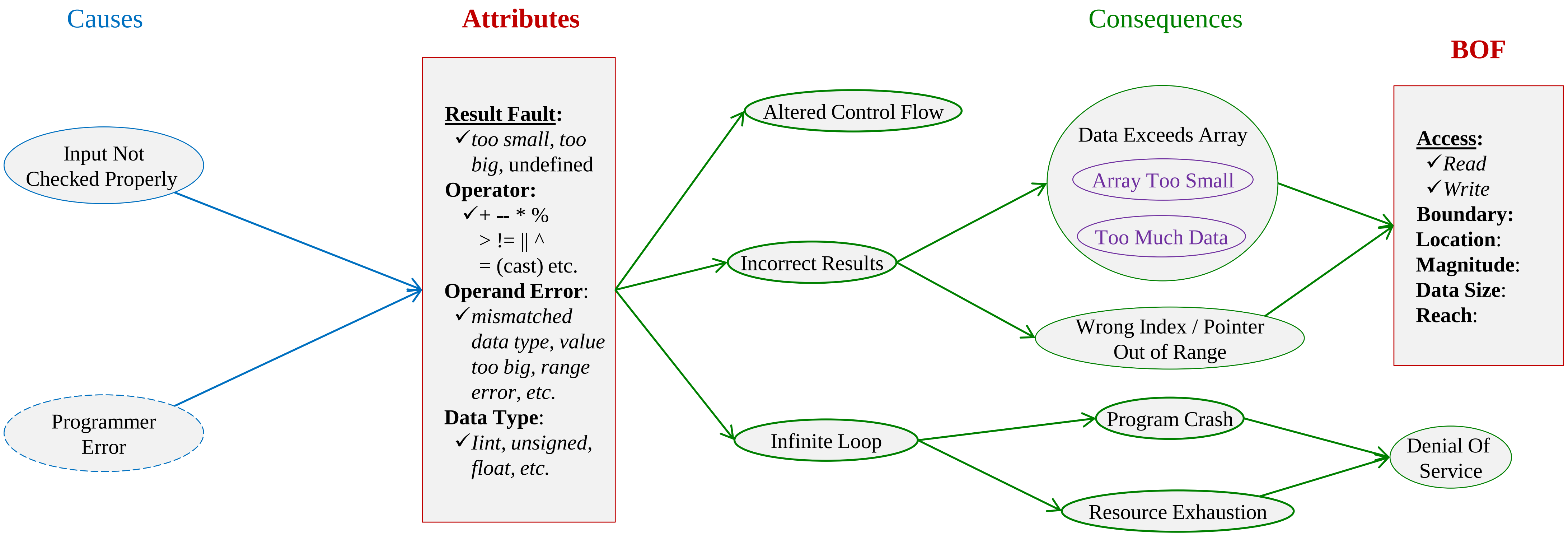
This is typically a relationship between operands of the operator, not just the characteristic of one operand.

FOP Attributes – Data Type

- Result Fault: value becomes small, large, undefined, etc.
- Operator: arithmetic (e.g. +, /, %, --), relational (e.g. <, !=), logical (&&, ||, and !), bitwise (&, |, ~, ^, >>, and <<), conditional (?:), assignment (e.g. =, +=), explicit conversion (cast), and argument passing in a function call.
- Operand Error: mismatched data types, value too big, range error, etc.
- Data Type: int, float, unsigned, etc.

This is additional description. For instance, mismatched data types could be `long int` *to* `short`*.*

FOP: Causes and Consequences

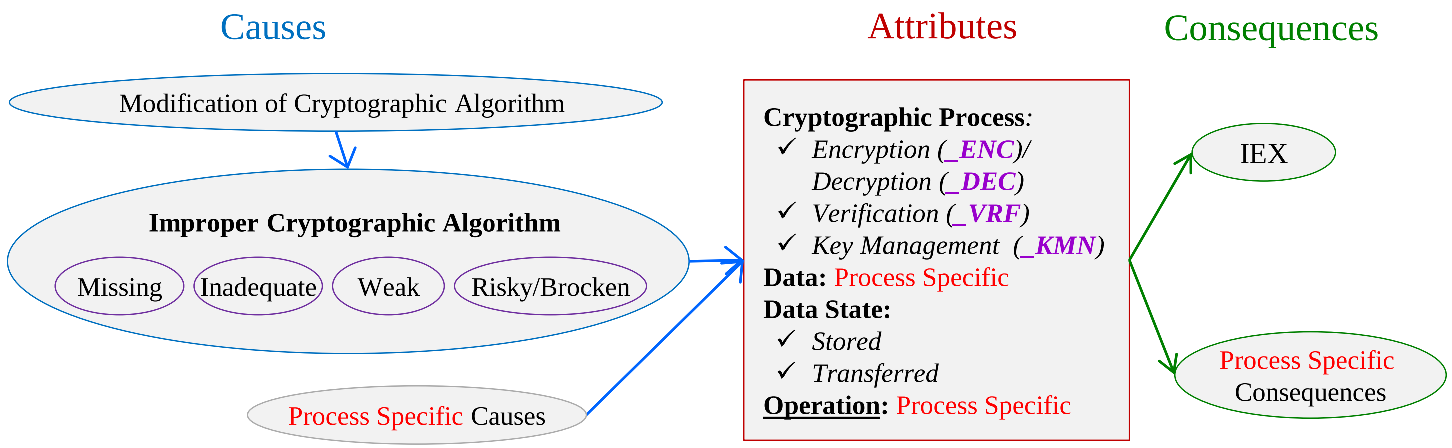


BF: Cryptography (CRY)

- Our Definition:
  The software does not properly encrypt/decrypt, verify, or manage keys for data (that has) to be securely stored/transferred.
  - _ENC: The software does not properly transform sensitive data (plaintext) into unintelligible form (ciphertext) using cryptographic algorithm and key(s).
  - _DEC: The software does not properly transform ciphertext into plaintext using cryptographic algorithm and key(s).
  - _VRF: The software does not properly sign message, check and prove sender, or assure message is not altered.
  - _KMN: The software does not properly generate, store, distribute, use, or destroy cryptographic keys (keying material).
- Related CWEs and SFPs:
  - ✓ CWEs related to CRY are CWE-311, CWE-325, CWE-327, CWE- 261, CWE-322, CWE-323, CWE-324, CWE-326, CWE-347, CWE-312 (incl. 313-318), CWE-256, CWE-257, CWE-295, CWE-296, CWE-321, CWE-329, CWE-780
  - ✓ Related SFPs are SFP 17.1 and SFP 17.2 under Primary Cluster: Cryptography.

BF: Cryptography (CRY)

CRY: The software does not properly encrypt/decrypt, verify, or manage keys for data (that has) to be securely stored/transferred.



Possible (attacks) → Man in the middle (MITM):
- ✓ Factoring keys
- ✓ Spoofing messages
- ✓ Replay instructions
- ✓ Brute force attack
- ✓ Timing attack
- ✓ CPA, CCA

Improper Cryptographic Algorithm (incl. Post-Quantum):
- Missing (at all, cryptographic step)
- Inadequate -- theoretically sound, but not strong enough for required level of protection
- Weak → prone to CPA, CCA
- Risky/Broken (as a whole, cryptographic step)
  (ex. DES now insufficient for many applications → replaced with AES)

Modification of Cryptographic Algorithm
- Remove//modify cryptographic step
- Add cryptographic step

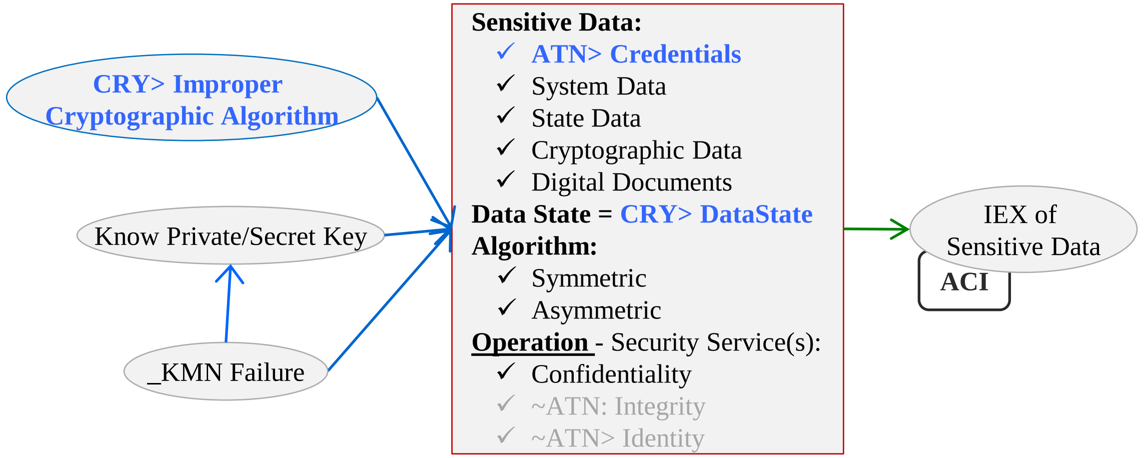
Cryptographic Process:
- ✓ _ENC/_DEC – needs _KMN
- ✓ _VRF - recursive _KMN, _VRF
- ✓ _KMN - recursive CRY, _KMN, _ENC, _VRF

Data State:
- ✓ Stored in:
  - File: ini, temp, configuration, log (server/debug/cleanup log), attachment to email, login buffer, executable, backup (.~bk), core dump, access control list, private data index
  - Directories (Web root, FTP root, CVS repository), Disk
  - Registry
  - Cookie (– incl. Persistent Cookies)
  - Source Code (incl. comments)
  - GUI, Environmental Variable
- ✓ Transferred:
  - Between Processes
  - Over Network

Cryptography (CRY): _ENC/ _DEC

_ENC: The software does not properly transform sensitive data (plaintext) into unintelligible form (ciphertext) using cryptographic algorithm and key(s).
_DEC: The software does not properly transform ciphertext into plaintext using cryptographic algorithm and key(s).

Causes Attributes Consequences



ATN> Credentials:
- ✓ Password, Key
- ✓ Token, PIV Card , Digital Certificate
- ✓ Biometrics

Sensitive Data (Secret - confidential):
- ✓ ATN> Credentials
- ✓ System Data (configurations, logs, Web usage)
- ✓ Cryptographic Data (hashes, keys, keying material)
- ✓ Digital Documents

Algorithm (Key Schema):
- Symmetric (Secret) Key – one key (e.g. Serpent, AES, IDEA, Blowfish)
- Asymmetric (Public) Key – two keys: public, private (e.g. D–H, DSA, RSA) – Key Size

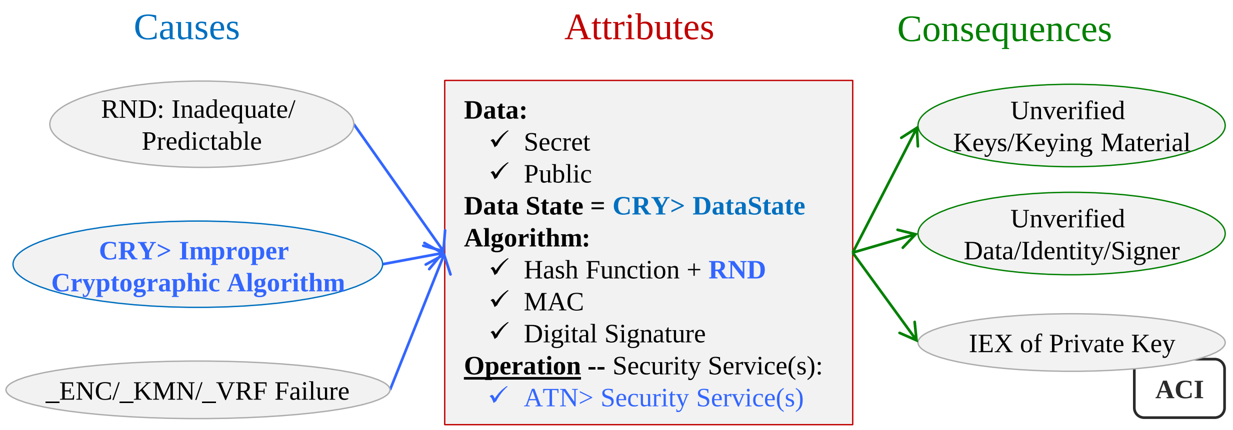
(Failed) Operation – Security Service(s):
- Confidentiality (sensitive data)
- ~Integrity Authentication*
- ~Identity Authentication*

* only for some specific modes of encryption

Cryptography (CRY): _VRF

_VRF: The software does not properly sign message, check and prove sender, or assure message is not altered.

"Check" is for identity authentication, "prove" is for signer non-repudiation, "alter" is for integrity authentication.



ATN> Security Service(s):

- Data Integrity Authentication
- Data Identity Authentication
- Signer Non-Repudiation

Data (confidential or not, needs verification):

- ✓ Secret (confidential) (Cryptographic hashes, secret keys, keying material)
- ✓ Public (Signed Contract, documents, public keys)

Algorithm(s):

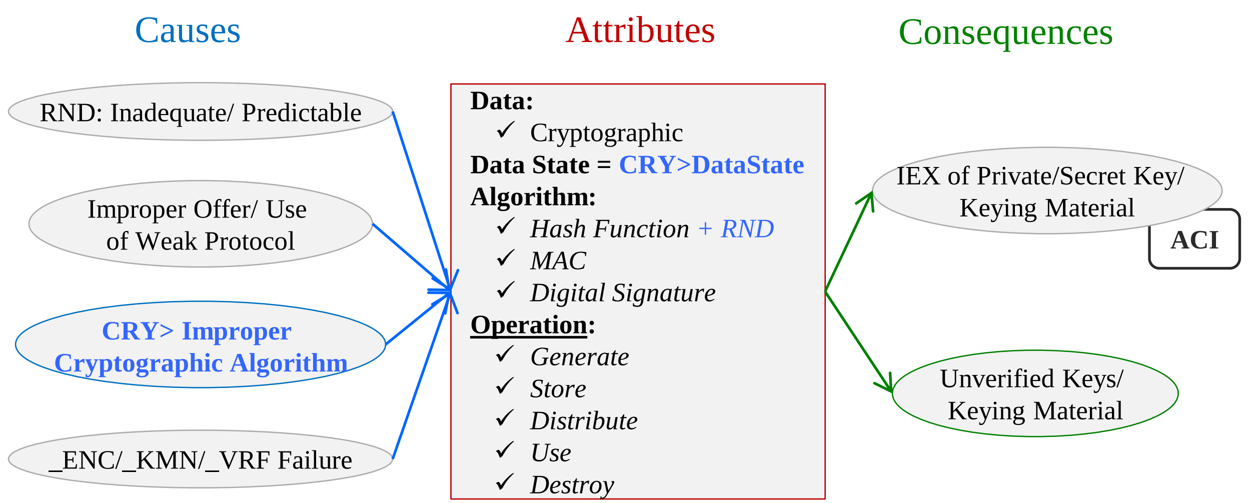
- ✓ Hash Function + RND (PRNG) - for Integrity Authentication → no Keys
- ✓ MAC (with unique keys per pair of users) -- recursive _KMN, _VRF → one Key (Symmetric/Secret Key Algorithm)
  for Integrity Authentication, Identity Authentication (algorithms for authentication code generation and message verification = generate key, sign ← get tag for key and message, verify by tag and key)
- ✓ Digital Signature -- recursive _KMN, _VRF → two keys (Asymmetric/Public Key Algorithm)
  for Integrity Authentication, Identity Authentication, Signer Non-Repudiation (algorithms for key generation, signature generation, and signature verification)

(Failed) Operation -- Security Service(s):

- ✓ Data/Keys Integrity Authentication
- ✓ Source Authentication: Identity Authentication, Signer Non-Repudiation

Cryptography (CRY): _KMN

_KMN: The software does not properly generate, store, distribute, use, or destroy cryptographic keys (keying material).

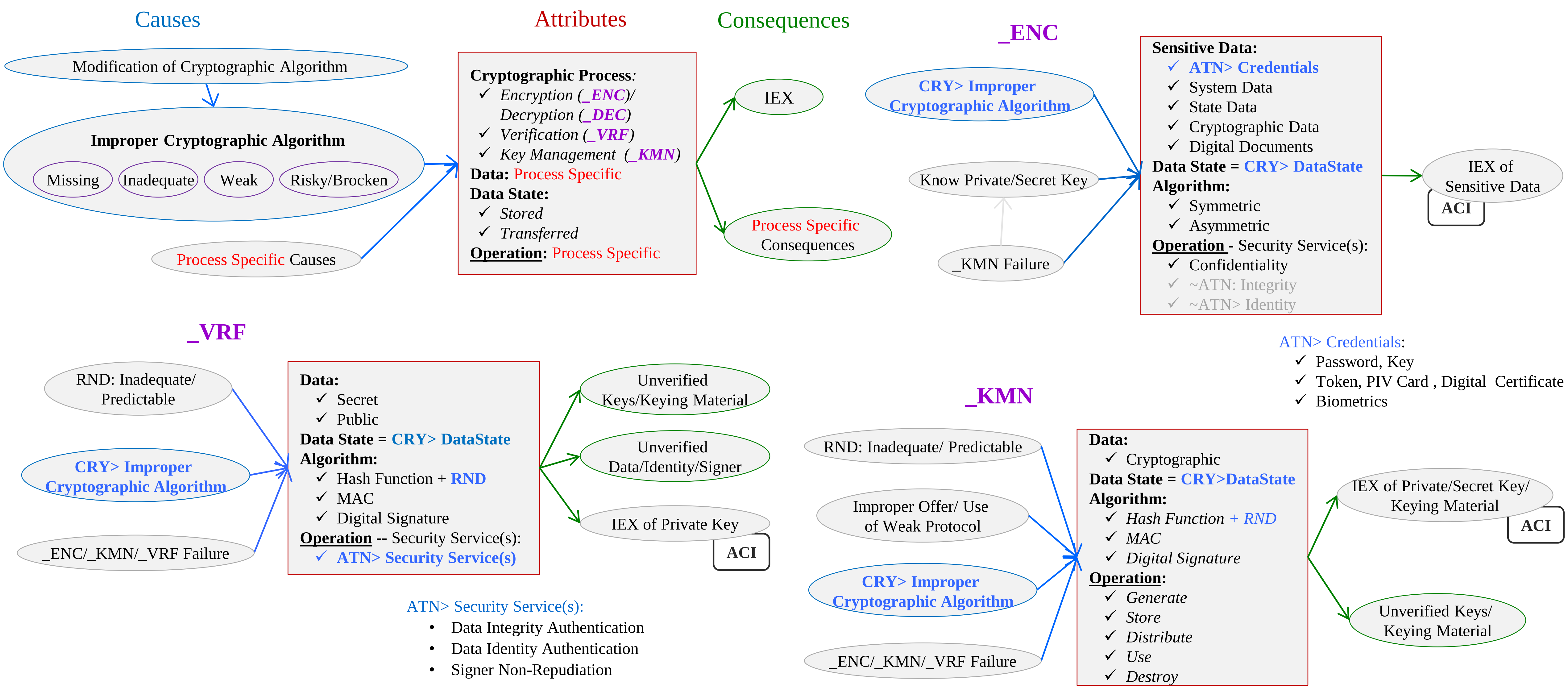


Data:
- ✓ Cryptographic Data (hashes, keys, keying material)

Algorithm (for key generation):
- Hash Function + RND (PRNG) → no Keys
- MAC -- recursive _VRF → one Key (Symmetric/Secret Key Algorithm)
- Digital Signature -- recursive _VRF → two keys (Asymmetric/Public Key Algorithm)

(Failed) Operation:
- Generate -- RND
- Store (Update, Recover) -- recursive CRY> State: Stored
- Distribute (Key establishment, transport, agreement, wrapping, encapsulation, derivation, confirmation; Shared secret creation) -- recursive _KMN, _ENC
- Use
- Destroy

BF: Cryptography (CRY)



CRY Exercises

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

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

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

CRY: Exercise 1 (FREAK)

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

CRY: Exercise 1 (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, and source code excerpts 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.” [17]

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.” [18]

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.” [19]

[17] The MITRE Corporation, CVE Common Vulnerabilities and Exposures, CVE-2015-0204.
[18] The MITRE Corporation, CVE Common Vulnerabilities and Exposures, CVE-2015-1637.
[19] The MITRE Corporation, CVE Common Vulnerabilities and Exposures, CVE-2015-1067.

CRY: Exercise 1 (FREAK) – Source Code

Client

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

Server

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

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

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

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

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

CRY: Exercise 1 (FREAK) – Analysis

The following analysis is based on information in [17-23].

- *ClientHello*: Client sends plaintext with no sensitive data (set of cipher suites client to use).
  → MITM intercepts that plaintext and changes it to request only Export RSA ciphersuite.
- *ServerHello*: If configured to offer Export RSA, server responds sending its SSL certificate.
- *Send ServerKeyExchange*: Server generates/retrieves 512-bit RSA key-pair, signs public key with its SSL certificate's private key to authenticate it to client, and sends the 512-bit key and signature.
  → MITM is watching exchange.
- *Receive ServerKeyExchange*: Client receives *ServerKeyExchange* (with weak Export RSA key). If there was no bug in OpenSSL, client should have said it did not ask for this and error out, shutting down the attack. However, due to the bug in OpenSSL the client would accept and use this weak key for the rest of the handshake.
- *Pre-Master Secret*: In both normal RSA and Export RSA, the Master Secret, used for symmetric encryption of all messages in the rest of the connection, is generated using Pre-Master Secret and two nonces (*ClientRandom* and *ServerRandom* sent in plaintext respectively in *ClientHello* and *ServerHello*). Client generates Pre-Master Secret as a random string, encrypts it using server's public key, and sends it to server. However, in Export RSA, server's public key is the 512-bit key from the *ServerKeyExchange* rather than from the server's SSL certificate.
  → MITM can factor that weak 512-bit public key to obtain the RSA private key. (Normal RSA public key is min 1024 bits to rander such factoring infeasible, but 512-bits public key can be factored using $100 of AWS computing time.)
  → With the RSA private key MITM can decrypt Pre-Master Secret, and then use it and the nonces to find the Master Secret (secret key used for symmetric encryption of transmitted data).
  → MITM now can decrypt, read, modify any massage between client and server, incl. passwords, credit cards info, ...

CRY: Exercise 1 (FREAK) – Analysis

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.

[23] includes: "OpenSSL clients would tolerate temporary RSA keys in non-export ciphersuites. It also had an option SSL_OP_EPHEMERAL_RSA which enabled this server side." and goes on to describe a response to the problem: "Remove both options as they are a protocol violation."

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

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

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

[23] GitHub, openssl, https://github.com/openssl/openssl/commit/ce325c60c74b0fa784f5872404b722e120e5cab0?diff=split

CRY: Exercise 1 (FREAK) – Solution

BF Description:

An inner _KMN CRY [1] leads to an inner _ENC CRY [2], which leads to an outer _ENC CRY [3].

[1] Inner _KMN CRY: Client-accepted improper offer of weak protocol (SSL with Export RSA) from MITM-tricked server, which generates 512-bit RSA key-pair, leads to IEX of transferred sensitive data (private key*).

[2] Inner _ENC CRY: Known private key for asymmetric algorithm (RSA) leads to failed confidentiality security service, decryption of transferred sensitive data (Pre-Master Secret**), and then IEX of other sensitive data (Master Secret***).

The inner CRYs only set up the secret key.

[3] Outer _ENC CRY: Known secret key (Master Secret) for symmetric algorithm leads to failed confidentiality security service, and decryption and IEX of transferred sensitive data (passwords, credit cards, etc.).

The outer CRY is the actual general data transmission.

CRY: Exercise 1 (FREAK) – Solution

* It is computationally feasible MITM to obtain the private key by factoring the public key for a 512-bit RSA key-pair. (In RSA, asymmetry is based on the practical difficulty of factoring the product of two large prime numbers.)
** Knowing the private key MITM can obtain the Pre-Master Secret by message decryption [1]."

*** Knowing Pre-Master Secret, MITM can generate Master Secret (Shared Secret Key).

Client-accepted: client code based on OpenSSL.

MITM-tricked server: server code does not properly restrict TLS state transitions.

MITM -- man in the middle.

Note: What is cool about this example is that the consequence from the first CRY causes the second CRY, which consequences cause the third CRY. The first inner _KMN CRY 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). The second inner CRY 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.

Next Steps: Bug Areas

- Software Weaknesses 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)
    - Integrity Authentication
    - Identity Authentication
    - Origin Authentication?
  - ✓ AUT (Authorization – often conflated with Access Control)
  - ✓ CRY (Cryptography)
    - _ENC (Encryption)
    - _VRF (Verification)
    - _KMN (Key Management)
  - ✓ RND (Randomization)
  - ✓ CIF (Control of Interaction Frequency)
  - ✓ 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) – Branching, Looping (Switch without default, Infinite loop,...)
  - ✓ INJ (Injection)
  - ✓ EXC (Exception handling)
    – Throw, Try, Catch
  - ✓ CON (Concurrency)
    – Deadlock, Starvation (unfair scheduling), Races, Locks, Synchronization, etc.

Upcoming: BF Data Classes

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

Questions

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