The NIST Bugs Framework (BF)

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

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

Irena Bojanova
Carlos Galhardo

Agenda

- Terminology:
  - Bug, Weakness
  - Vulnerability
  - Failure
- Existing Repositories:
  - CWE
  - CVE
  - NVD
- The Bugs Framework (BF)
  - Goals
  - Features
- Example – Heartbleed
- Potential Impacts

Terminology

Bug, Weakness, Vulnerability, Failure

- Software Bug:
  - A coding error
  - Needs to be fixed
- Software Weakness – difficult to define:
  - Caused by a bug or ill-formed data
  - Weakness Type – a meaningful notion!
- Software Vulnerability:
  - An instance of a weakness type that leads to a security failure
  - May have several underlying weaknesses
- Security failure:
  - A violation of a system security requirement

Existing Repositories

Commonly Used Repositories

- Weaknesses:
  CWE – Common Weakness Enumeration

- Vulnerabilities:
  CVE – Common Vulnerabilities and Exposures
  → over 18 000 documented in 2020

- Linking weaknesses to vulnerabilities – CWEs to CVEs:
  NVD – National Vulnerabilities Database

Repository Problems

1. Imprecise Descriptions – CWE & CVE
2. Unclear Causality – CWE & CVE
3. Gaps in Coverage – CWE
4. Overlaps in Coverage – CWE

Problem #1: Imprecise Descriptions

- Example:

  CWE-502: Deserialization of Untrusted Data:

  The application deserializes untrusted data without *sufficiently* verifying that the resulting data will be valid.

  - Unclear what "*sufficiently*" means,
  - "verifying that data is valid" is also confusing

Problem #2: Unclear Causality

- Example:

  CVE-2018-5907
  Possible buffer overflow in msm_adsp_stream_callback_put due to lack of input validation of user-provided data that leads to integer overflow in all Android releases (Android for MSM, Firefox OS for MSM, QRD Android) from CAF using the Linux kernel.

  → the NVD label is CWE-190

  While the CWEs chain is:
  CWE-20 → CWE-190 → CWE-119

Problems #3, #4: Gaps/Overlaps in Coverage

- Example:

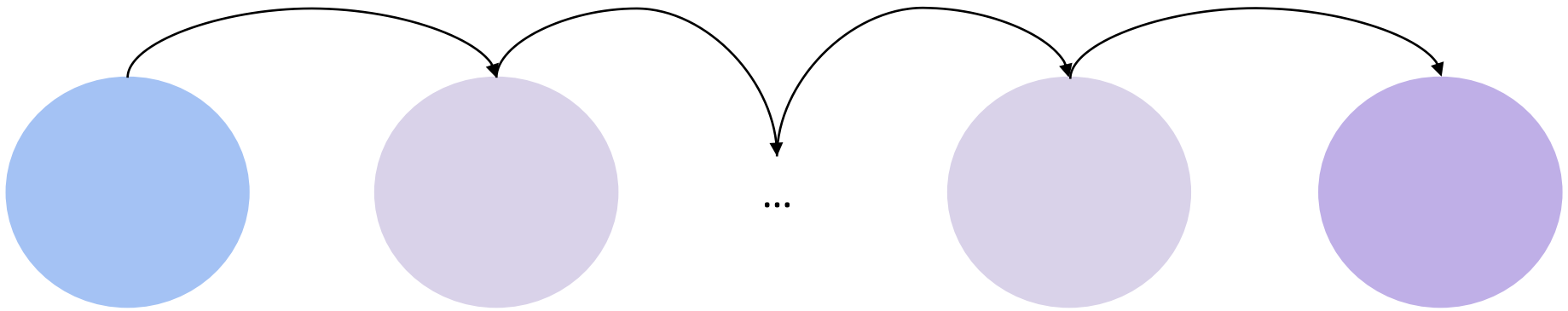
  CWEs coverage of buffer overflow by:
  - ✓ Read/ Write
  - ✓ Over/ Under
  - ✓ Stack/ Heap

	Over	Under	Either End		Stack	Heap
Read	CWE-127	CWE-126	CWE-125		✦	✦
Write	CWE-124	CWE-120	CWE-123 CWE-787 ✦		CWE-121	CWE-122
Read/ Write	CWE-786	CWE-788	✦		✦	✦

The Bugs Framework (BF)

BF Goals

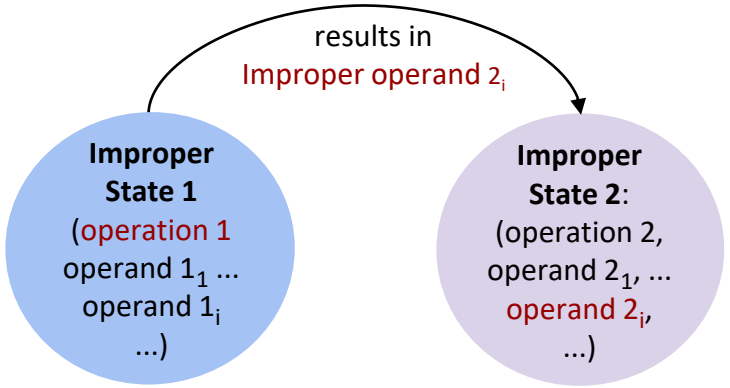
1. Solve the problems of imprecise descriptions and unclear causality



2. Solve the problems of gaps and overlaps in coverage

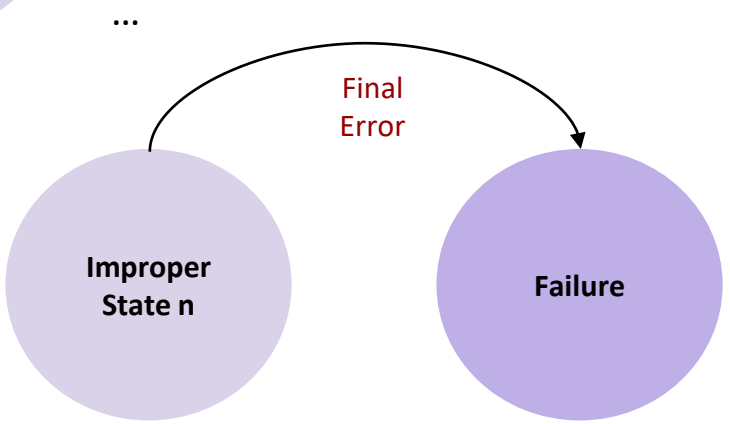
BF Features – Clear Causal Descriptions

- BF describes a bug/weakness as:
  - An improper state
    and
  - Its transition



- Improper State –
  a tuple (`operation`, `operand`$_1$, `...` , `operand`$_n$)
  , where at least one element is improper

- Transition –
  the result of the `operation` over the `operands`



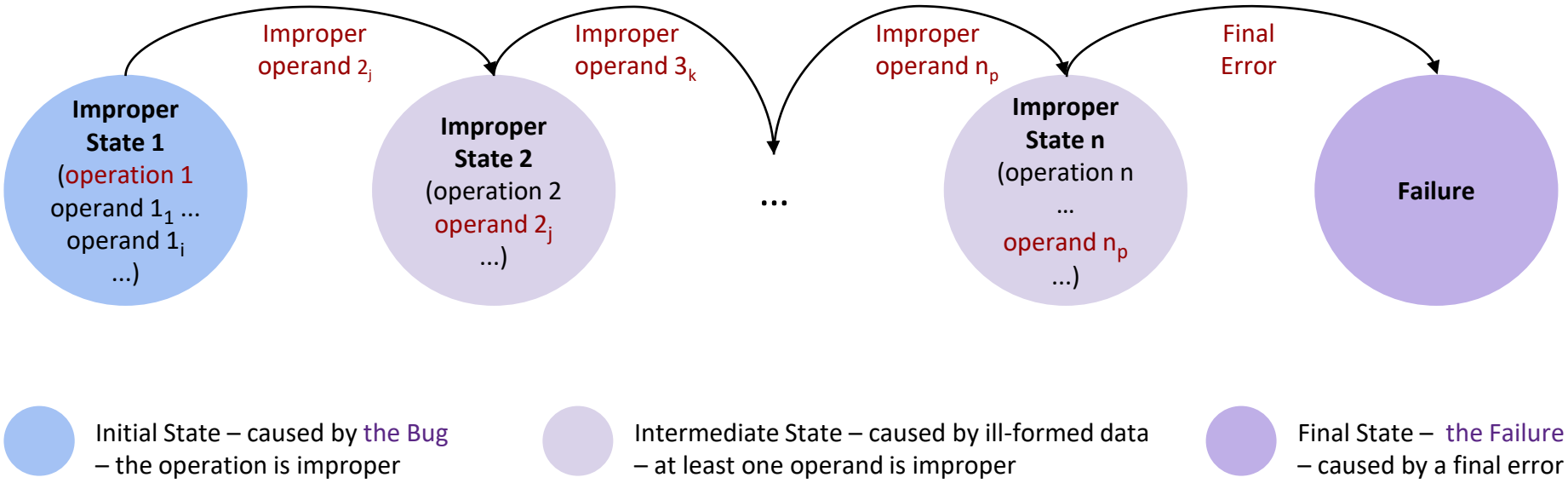
Initial State – caused by the Bug
– the operation is improper

Intermediate State – caused by ill-formed data
– at least one operand is improper

Final State – the Failure
– caused by a final error

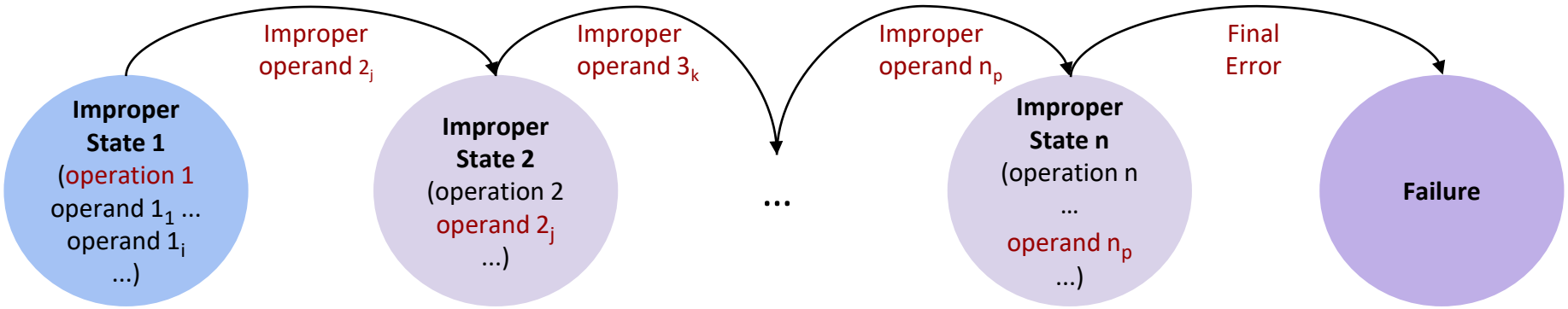
BF Features – Chaining Weaknesses

- BF describes a vulnerability as:
  - A chain of improper states and their transitions
  - States change until a failure is reached



BF Features – Causes and Consequences

- How to find the Bug?
- Go backwards by operand until an operation is a cause

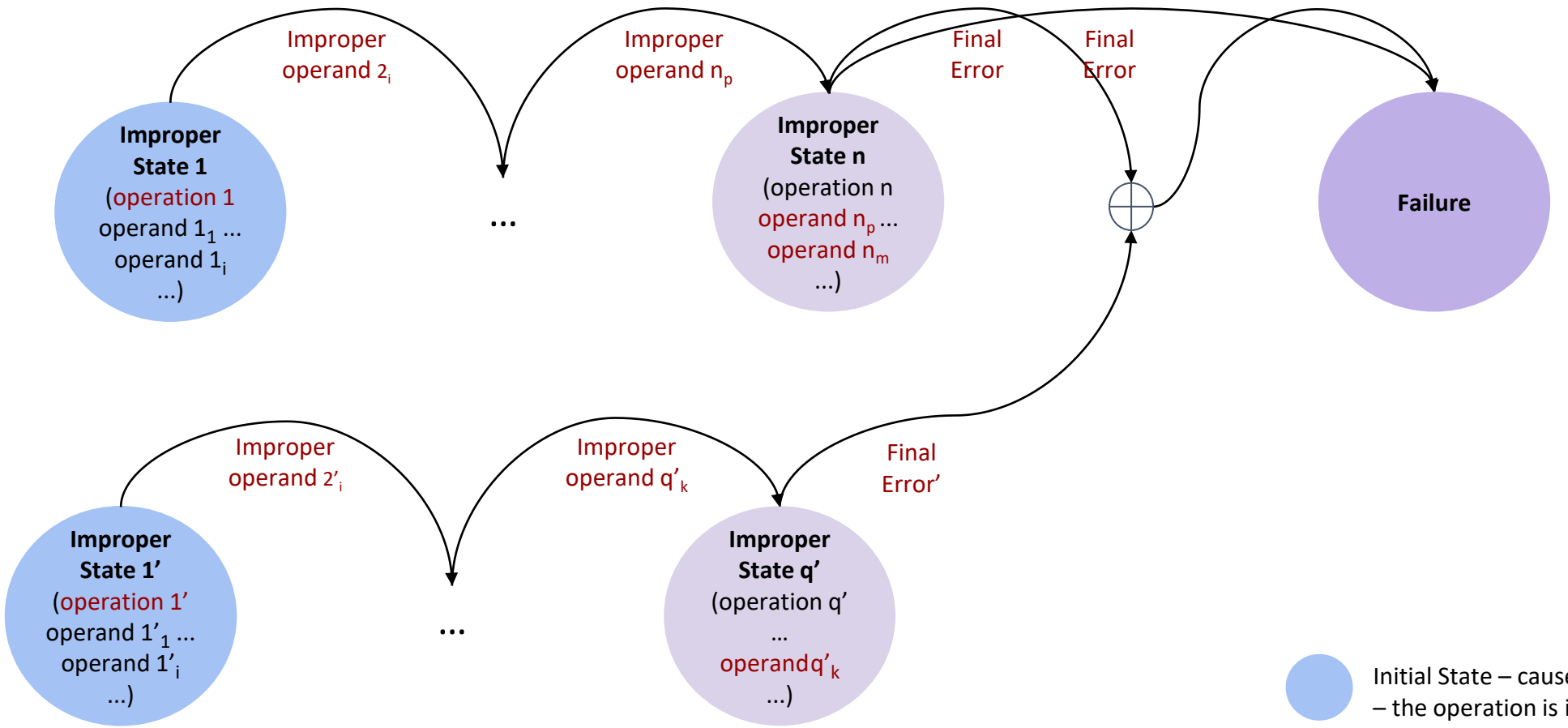


Initial State – caused by the Bug
– the operation is improper

Intermediate State – caused by ill-formed data
– at least one operand is improper

Final State – the Failure
– caused by a final error

BF Features – Converging Vulnerabilities



BF Features – Classification

- BF Class – a taxonomic category of a weakness type, defined by:
  - A set of operations
  - All valid cause → consequence relations
  - A set of attributes

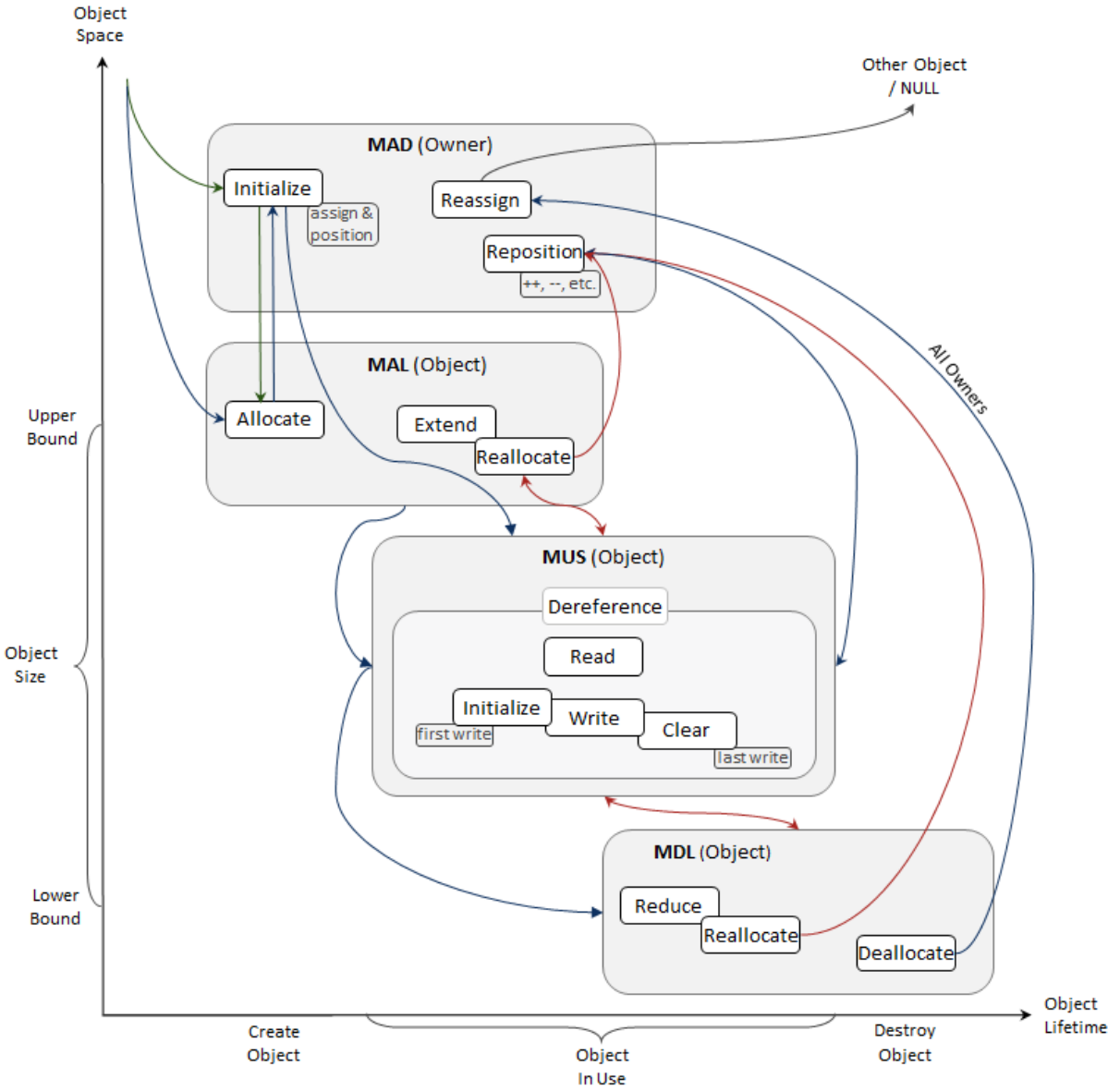
BF – Bugs Models

- Example:

  The BF Memory Bugs Model:

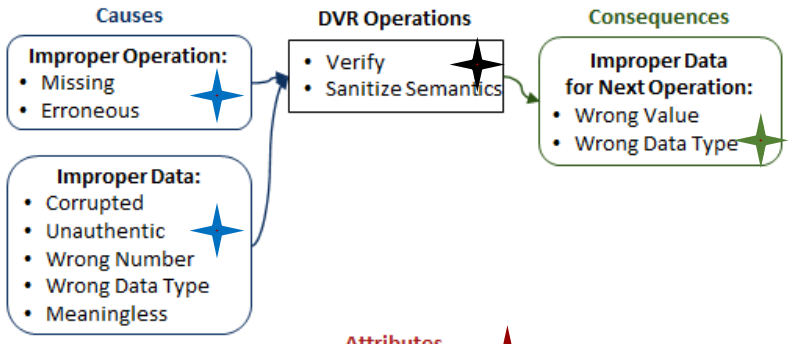
  - Four phases, corresponding to the BF memory bugs classes: MAD, MAL, MUS, MDL

  - Memory operations flow



BF Classes – Examples

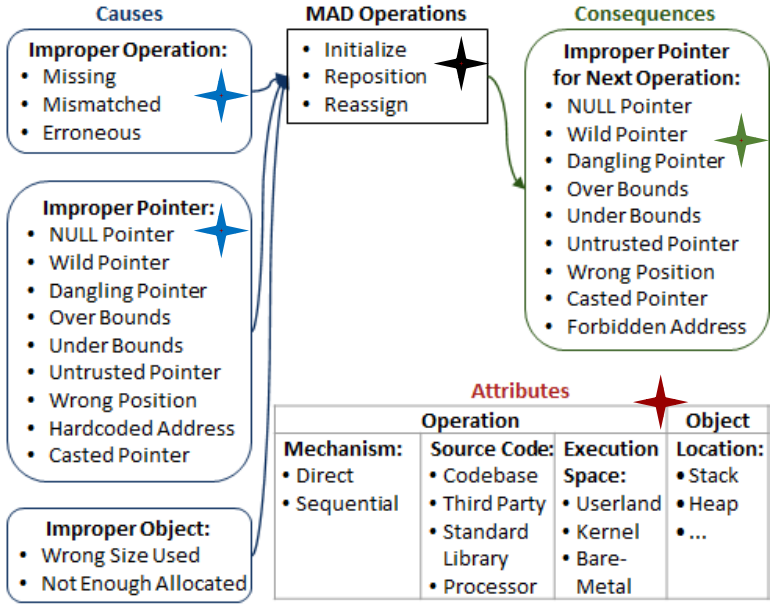
Data Verification Bugs (DVR)



Attributes

Operation			Data	
Mechanism: • Range • Is Null • Safe List • Unsafe List • Business Logic	**Source Code:** • Codebase • Third Party • Standard Library • Processor	**Execution Space:** • Userland • Kernel • Bare-Metal	**Location:** • User Entered • Stored • Transferred • In Use	**Side:** • Client • Server

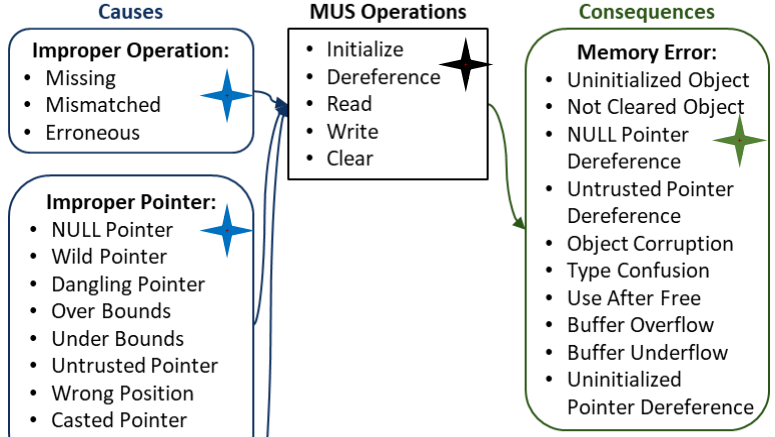
Memory Addressing Bugs (MAD)



Attributes

Operation			Object
Mechanism: • Direct • Sequential	**Source Code:** • Codebase • Third Party • Standard Library • Processor	**Execution Space:** • Userland • Kernel • Bare-Metal	**Location:** • Stack • Heap • ...

Memory Use Bugs (MUS)



Attributes

Operation			Pointer	Object
Mechanism: • Direct • Sequential	**Source Code:** • Codebase • Third Party • Standard Library • Processor	**Execution Space:** • Userland • Kernel • Bare-Metal	**Span:** • Little • Moderate • Huge	**Location:** • Stack • Heap • ...

BF – Defined

- BF is a …
  - ➢ Structured
  - ➢ Complete
  - ➢ Orthogonal
  - ➢ Language independent

classification of software bugs and weaknesses

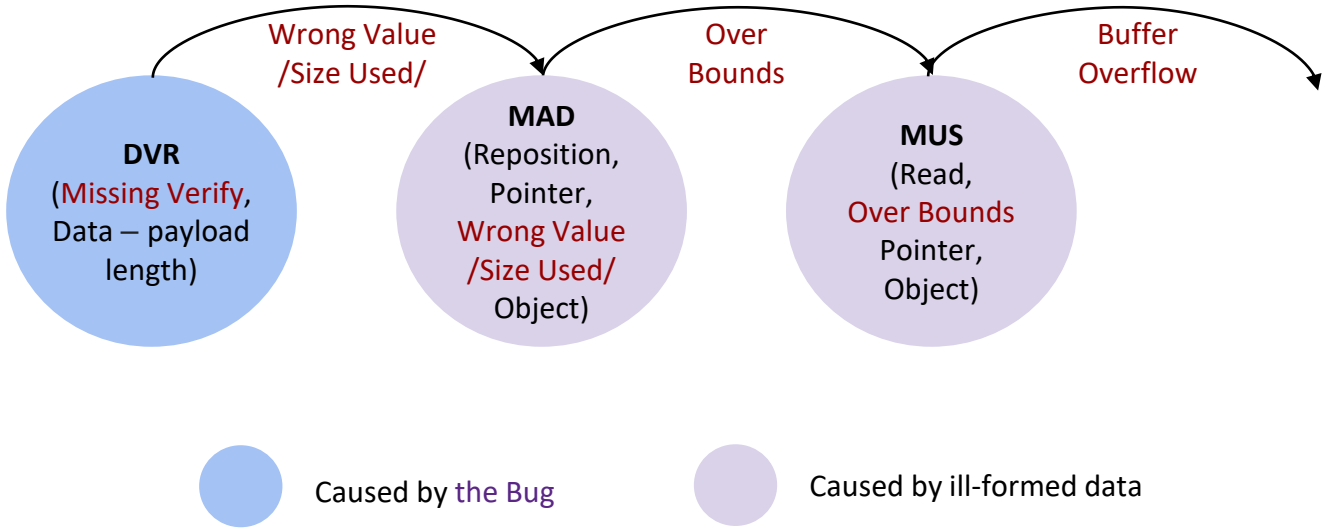
BF Example – Description of Heartbleed

Heartbleed (CVE-2014-0160)

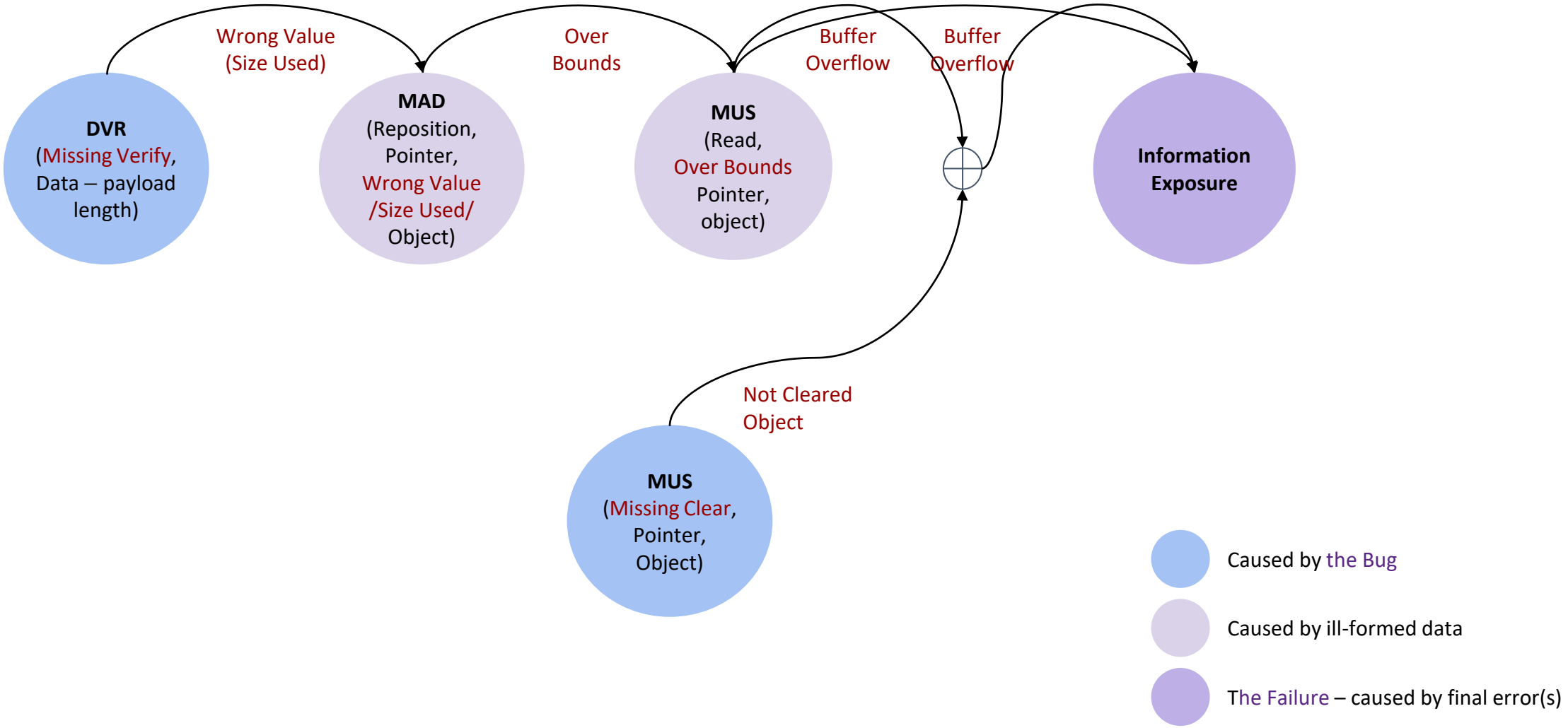
CVE-2014-0160 The (1) TLS and (2) DTLS implementations in OpenSSL 1.0.1 before 1.0.1g do not properly handle Heartbeat Extension packets, which allows remote attackers to obtain sensitive information from process memory via crafted packets that trigger a buffer over-read, as demonstrated by reading private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

```
1448 dtls1_process_heartbeat(SSL *s)
1449   {
1450   unsigned char *p = &s->s3->rrec.data[0], *pl;
1451   unsigned short hbtype;
1452   unsigned int payload;
1453   unsigned int padding = 16; /* Use minimum padding */
1454
1455   /* Read type and payload length first */
1456   hbtype = *p++;
1457   n2s(p, payload);
1458   pl = p;
...
1465   if (hbtype == TLS1_HB_REQUEST)
1466     {
1467     unsigned char *buffer, *bp;
...
1470     /* Allocate memory for the response, size is 1 byte
1471     * message type, plus 2 bytes payload, plus
1472     * payload, plus padding
1473     */
1474     buffer = OPENSSL_malloc(1 + 2 + payload + padding);
1475     bp = buffer;
1476
1477     /* Enter response type, length and copy payload */
1478     *bp++ = TLS1_HB_RESPONSE;
1479     s2n(payload, bp);
1480     memcpy(bp, pl, payload);
```

```
/* Naive implementation of memcpy
void *memcpy (void *dst, const void *src, size_t n)
{
    size_t i;          payload
    for (i=0; i<n; i++)
        *(char *) dst++ = *(char *) src++;
    return dst;
}                      bp                pl
```



Clear Causality in Heartbleed



BF Description of Heartbleed

Cause → **DVR Operation** → **Consequence**

Cause	DVR Operation	Consequence
Improper Operation: Missing	Verify	**Improper Data:** Wrong Value (`payload_size`)

Attributes

Operation			Data	
Mechanism: • Range	Source Code: • Codebase (`d1_both.c` and `tl_lib.c`)	Execution Space: • Userland	Location: • Transferred	Side: • Server

Cause → **MAD Operation** → **Consequence**

Cause	MAD Operation	Consequence
Improper Object: Wrong Size Used (for `s→s3→rrec.data[0]`)	Reposition	**Improper Pointer:** Over Bounds

Attributes

Operation			Object
Mechanism: • Sequential	Source Code: • Codebase (`d1_both.c` and `tl_lib.c`)	Execution Space: • Userland	Location: • Heap

Cause → **MUS Operation** → **Consequence**

Cause	MUS Operation	Consequence
Improper Operation: Missing	Clear	**Memory Error:** Not Cleared Object

Attributes

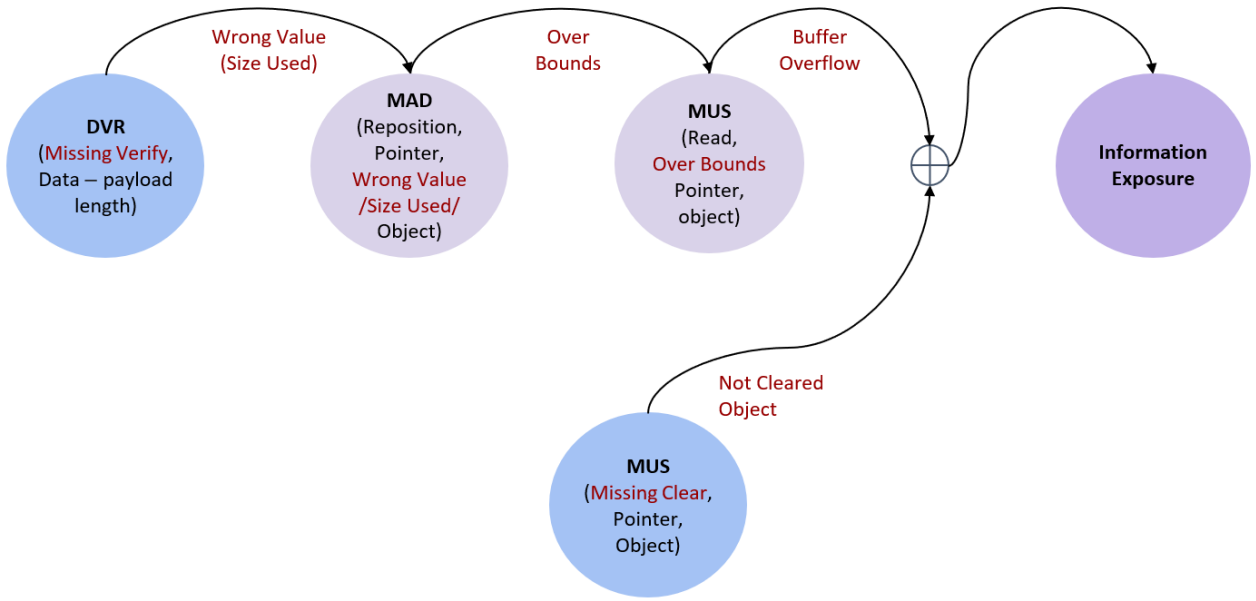
Operation			Pointer	Object
Mechanism: • Sequential	Source Code: • Codebase	Execution Space: • Userland	Span: • Huge	Location: • Heap

Cause → **MUS Operation** → **Consequence**

Cause	MUS Operation	Consequence
Improper Pointer: Over Bounds (for `s→s3→rrec.data[0]`)	Read	**Memory Error:** Buffer Overflow

Attributes

Operation			Pointer	Object
Mechanism: • Sequential	Source Code: • Codebase (`d1_both.c` and `tl_lib.c`)	Execution Space: • Userland	Span: • Huge	Location: • Heap

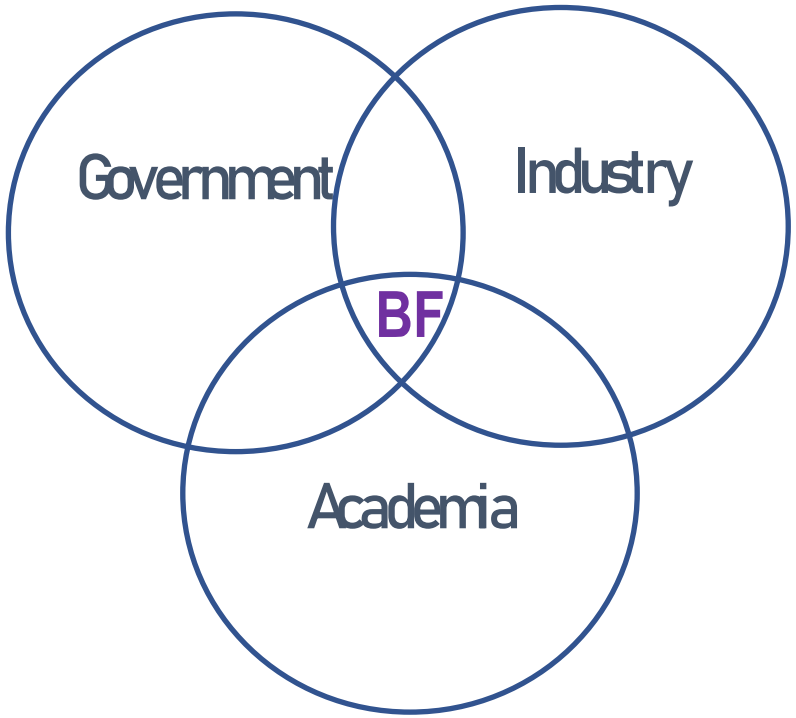


BF – Potential Impact

BF – Potential Impacts

- Allow precise communication
  about software bugs and weaknesses

- Help identify exploit mitigation techniques



Questions

Questions

Irena Bojanova: irena.bojanova@nist.gov

Carlos Galhardo: cegalhardo@inmetro.gov.br

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