Bugs Framework (BF)
Formalizing Cybersecurity Weaknesses and Vulnerabilities

Office of the National Cyber Director (ONCD)

April 4, 2024

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

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

Agenda

- **Introduction**
  - CWE, CVE, NVD
  - BF Approach
  - BF Security Concepts

- **BF**
  - Bugs Models
  - Weakness Taxonomies
  - Vulnerability Models
  - Formal Language

- **BF Datasets**
  - BFCWE
  - BCVE

- **BF Vulnerability Classification Model**

- **Potential Impacts**

Introduction

Current State of the Art

- Weaknesses

  CWE – Common Weakness Enumeration   https://cwe.mitre.org/

- Vulnerabilities

  CVE – Common Vulnerabilities and Exposures   https://cve.mitre.org/

- Assigning weaknesses to vulnerabilities – CWEs to CVEs

  NVD – National Vulnerabilities Database   https://nvd.nist.gov/

Repository Challenges

1. Imprecise descriptions
2. Unclear causality
3. Gaps in coverage
4. Overlaps in coverage
5. Wrong NVD assignments
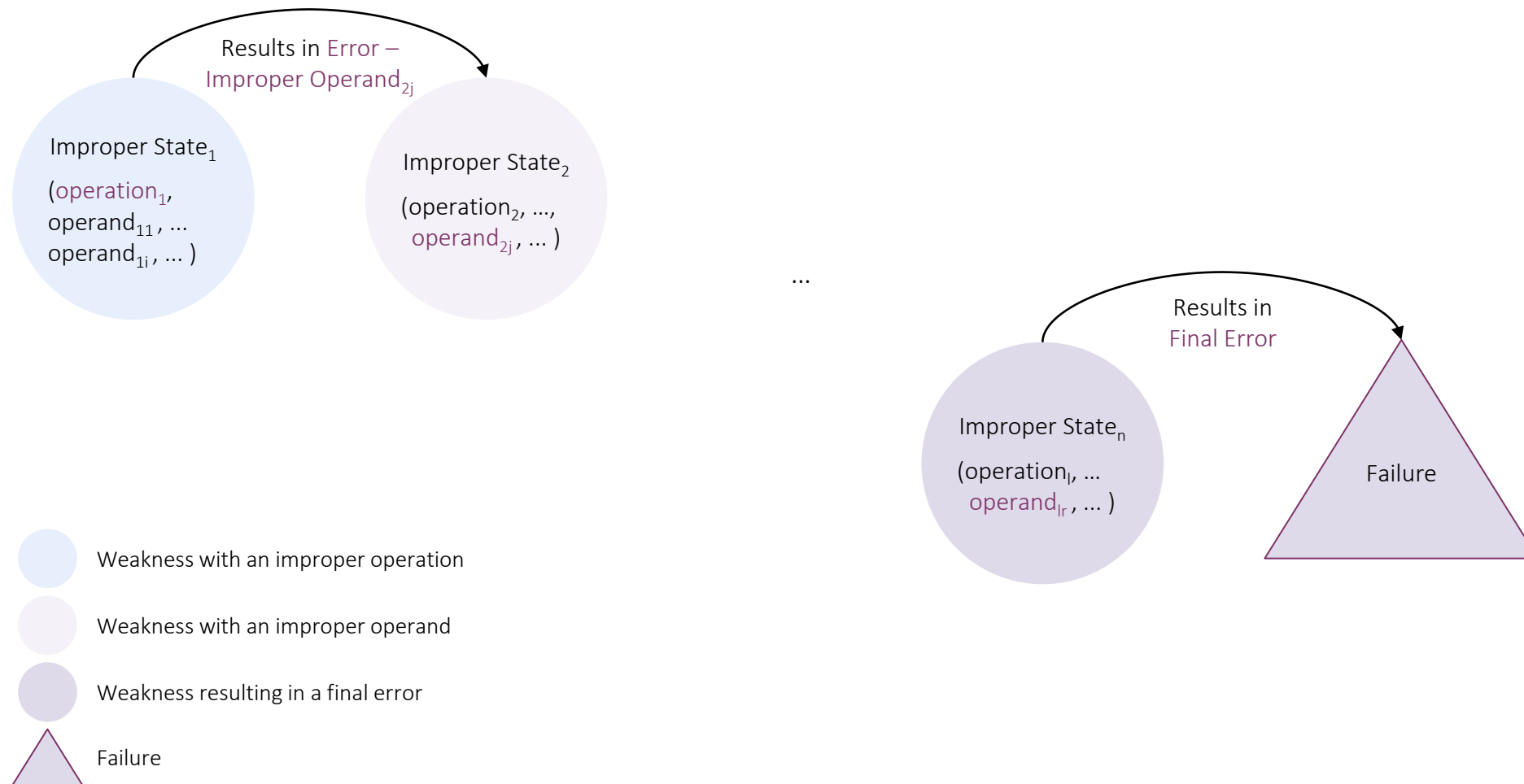6. No tracking methodology
7. No tools

★ Focus is on descriptions

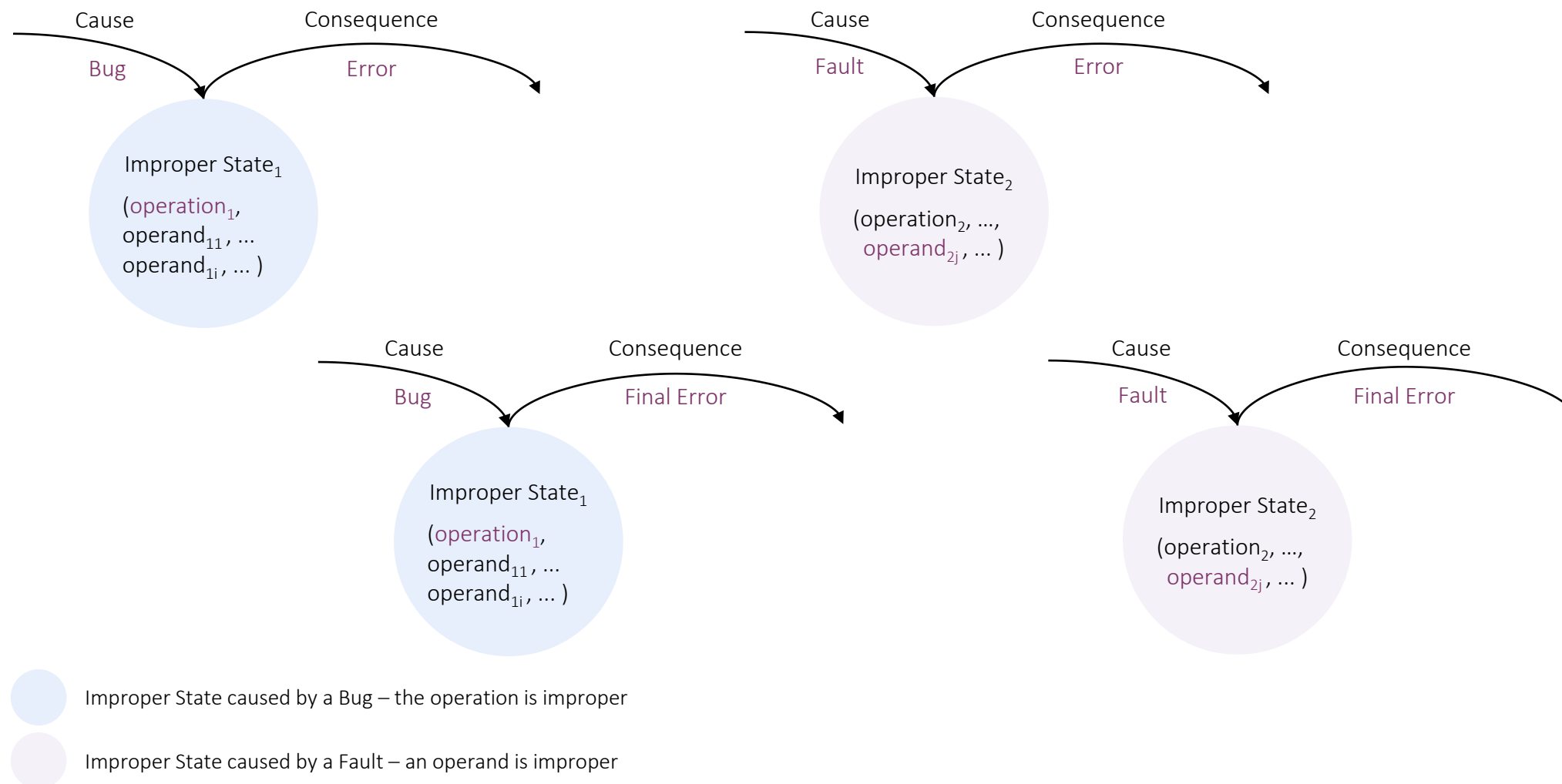| Repository Challenges	Imprecise Descriptions	Unclear Causality	Gaps in Coverage	Overlaps in Coverage	Wrong CVE to CWE mapping	No Tracking Methodology	No Tools
CWE	✓	✓	✓	✓		✓	✓
CVE	✓	✓				✓	✓
NVD	✓	✓			✓	✓	✓

BF Approach

BF is a classification of security bugs and related faults, featuring a formal language for unambiguous specification of weaknesses and underlined by them vulnerabilities.

- Bugs and faults – as weakness causes

- Errors and final errors – as weakness consequences

- BF formal language – based on:
  - Weakness taxonomies
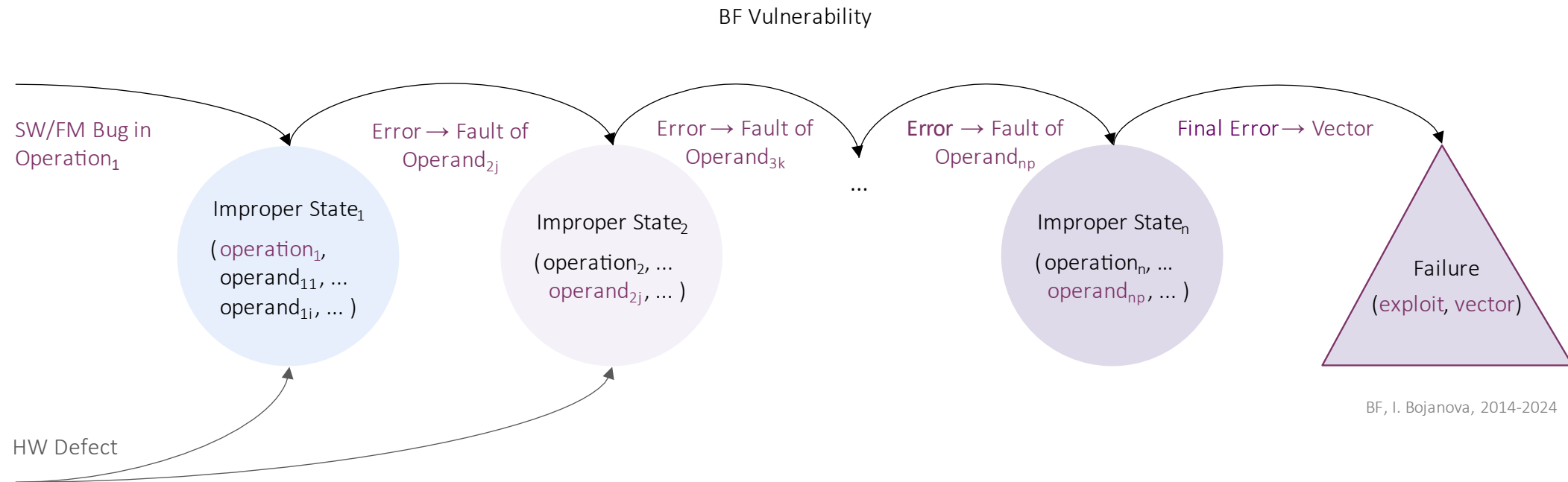 - Bugs models
  - Vulnerability models

BF Weakness

Results in Error – Improper Operand$_{2j}$

Improper State$_1$

(operation$_1$,
operand$_{11}$, …
operand$_{1i}$, …)

Improper State$_2$

(operation$_2$, …,
operand$_{2j}$, …)

…

Results in Final Error

Improper State$_n$

(operation$_l$, …
operand$_{lr}$, …)

Failure

Weakness with an improper operation

Weakness with an improper operand

Weakness resulting in a final error

Failure

BF Weakness States

Cause — Bug   **Consequence** — Error

Improper State$_1$
(operation$_1$,
operand$_{11}$, ...
operand$_{1i}$, ...)

Cause — Fault   **Consequence** — Error

Improper State$_2$
(operation$_2$, ...,
operand$_{2j}$, ...)

Cause — Bug   **Consequence** — Final Error

Improper State$_1$
(operation$_1$,
operand$_{11}$, ...
operand$_{1i}$, ...)

Cause — Fault   **Consequence** — Final Error

Improper State$_2$
(operation$_2$, ...,
operand$_{2j}$, ...)

Improper State caused by a Bug – the operation is improper

Improper State caused by a Fault – an operand is improper

BF Vulnerability

BF Vulnerability

SW/FM Bug in
Operation$_1$

$Error \rightarrow Fault\ of\ Operand_{2j}$

$Error \rightarrow Fault\ of\ Operand_{3k}$

$\textbf{Error} \rightarrow \textbf{Fault of}\ \textbf{Operand}_{np}$

$\textbf{Final Error} \rightarrow \textbf{Vector}$

Improper State$_1$

(operation$_1$,
operand$_{11}$, ...
operand$_{1i}$, ...)

Improper State$_2$

(operation$_2$, ...
operand$_{2j}$, ...)

...

Improper State$_n$

(operation$_n$, ...
operand$_{np}$, ...)

Failure
(exploit, vector)

BF, I. Bojanova, 2014-2024

HW Defect

Improper State: an *(operation, operand$_1$, ..., operand$_n$)* tuple with at least one improper element

⤴ Chaining

Initial State − caused by a Bug

Propagation State − caused by a Fault

Final State − supplies an Exploit Vector

Failure − result of the exploit of the vector supplied by the Final Error

BF Bugs Detection

BF Bug Identification
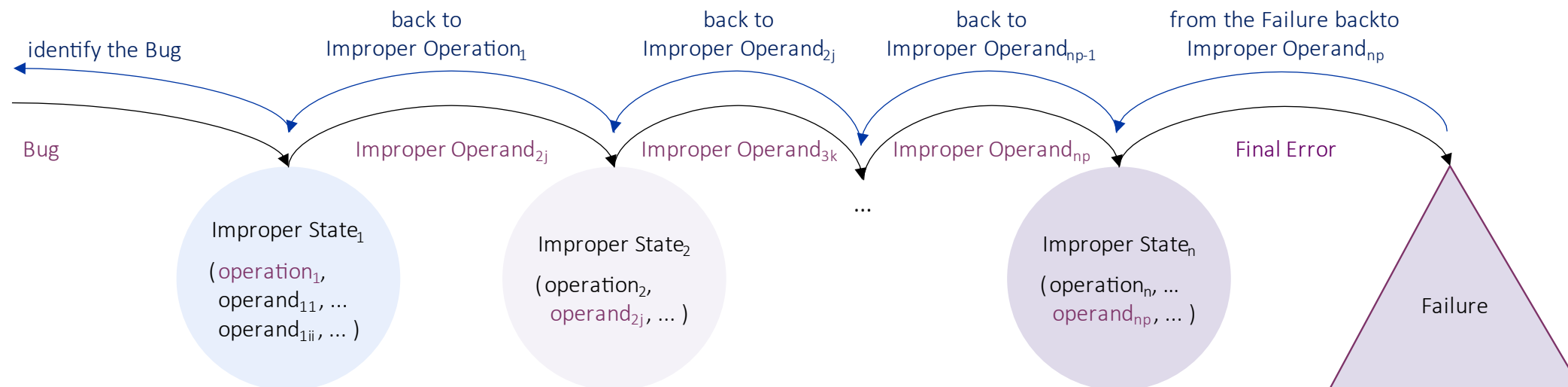
BF, I. Bojanova, 2014-2024

Improper State: an *(operation, operand$_1$, …, operand$_n$)* tuple with at least one improper element

↷ Chaining   ↶ Backtrack to previous State

Initial State – caused by a Bug

Propagation State – caused by a Fault

Final State – results in an Exploitable Error

Failure – caused by exploitation of the Final Error
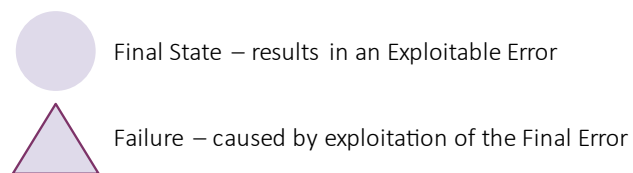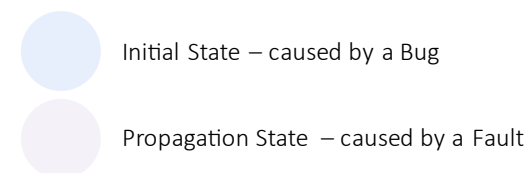
BF Security Concepts

Bug/Fault – relates to Execution Phase:
- Operations
- Input Operands
- Output Results

- ## Security Bug
  - Code or specification defect
  - May result from a hardware defect
  - May resurface by configuration/environment
- ## Fault
  - Name, data, type, address, or size error
  - Could be from a Bug or induced by a hardware defect
- ## Error
  - From bug or fault
  - Propagates to another fault
- ## Security Final Error
  - From bug or fault
  - Undefined system behavior

- ## Security Weakness
  - ```
    (bug, operation, error)
    (fault, operation, error)
    (bug, operation, final error)
    (fault, operation, final error)
    ```

- ## Security Vulnerability
  - Chain of weaknesses
  - Bug → Error/Fault → … → Final Error

- ## Security Failure
  - Violation of system security requirement
    - Information Exposure (IEX)
    - Data Tempering (TPR)
    - Denial of Service (DoS)
    - Arbitrary Code Execution (ACE)

BF Security Concepts Model

NIST

BF Software Security Concepts Model
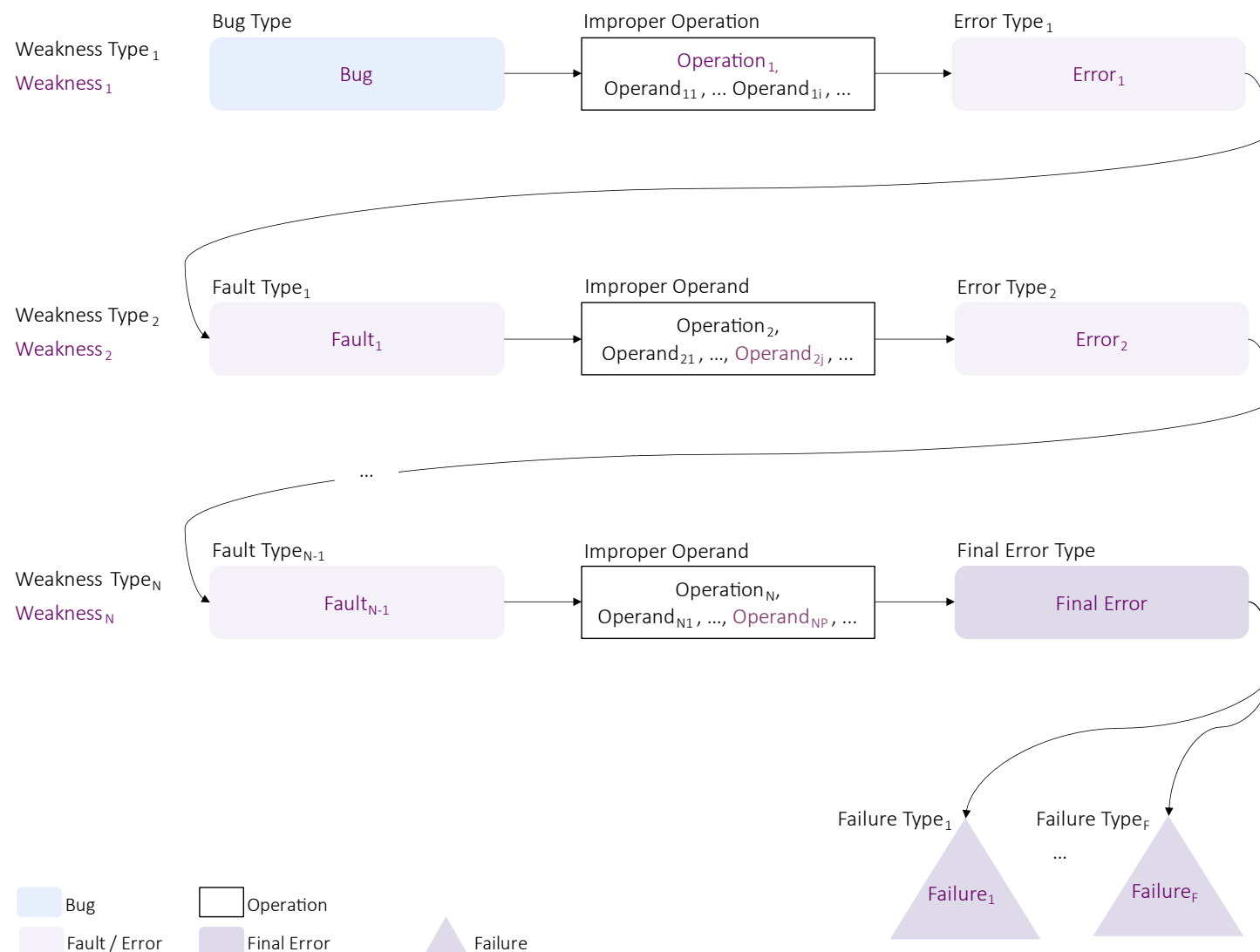
Operation$_1$ has a Bug and results in Error 1, which becomes Fault$_1$ for Operation$_2$, leading to Error$_2$.

The chain goes on, until the last operation results in a Final Error, leading to a Failure.

Legend:
- Bug
- Operation
- Fault / Error
- Final Error
- Failure

BF, I. Bojanova 2014-2024

I. Bojanova, 2024

BF Bugs Models

BF Memory (_MEM) Bugs Models

- Identify Secure Code Principles:
  - Memory Safety

BF Input/Output Model (_INP) Bugs Model

BF Input/Output Check Bugs Model

BF, I. Bojanova, 2020-2024
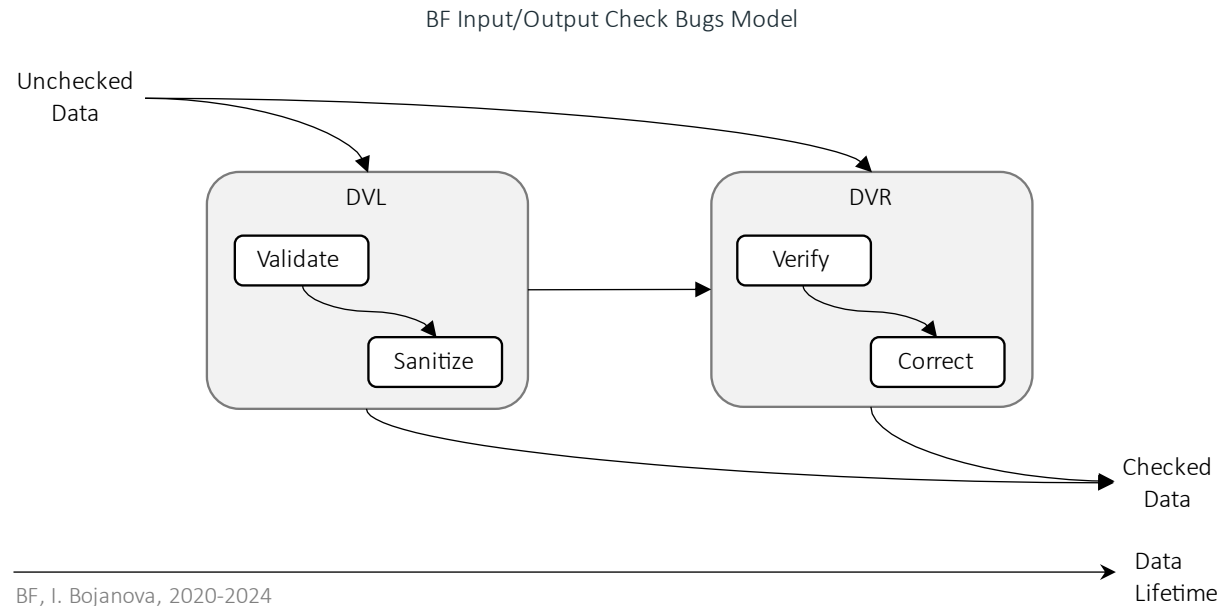
- Identify Secure Code Principles:
  - Input/Output Safety

BF Data Type (_DAT) Bugs Model

BF Data Type Bugs Model
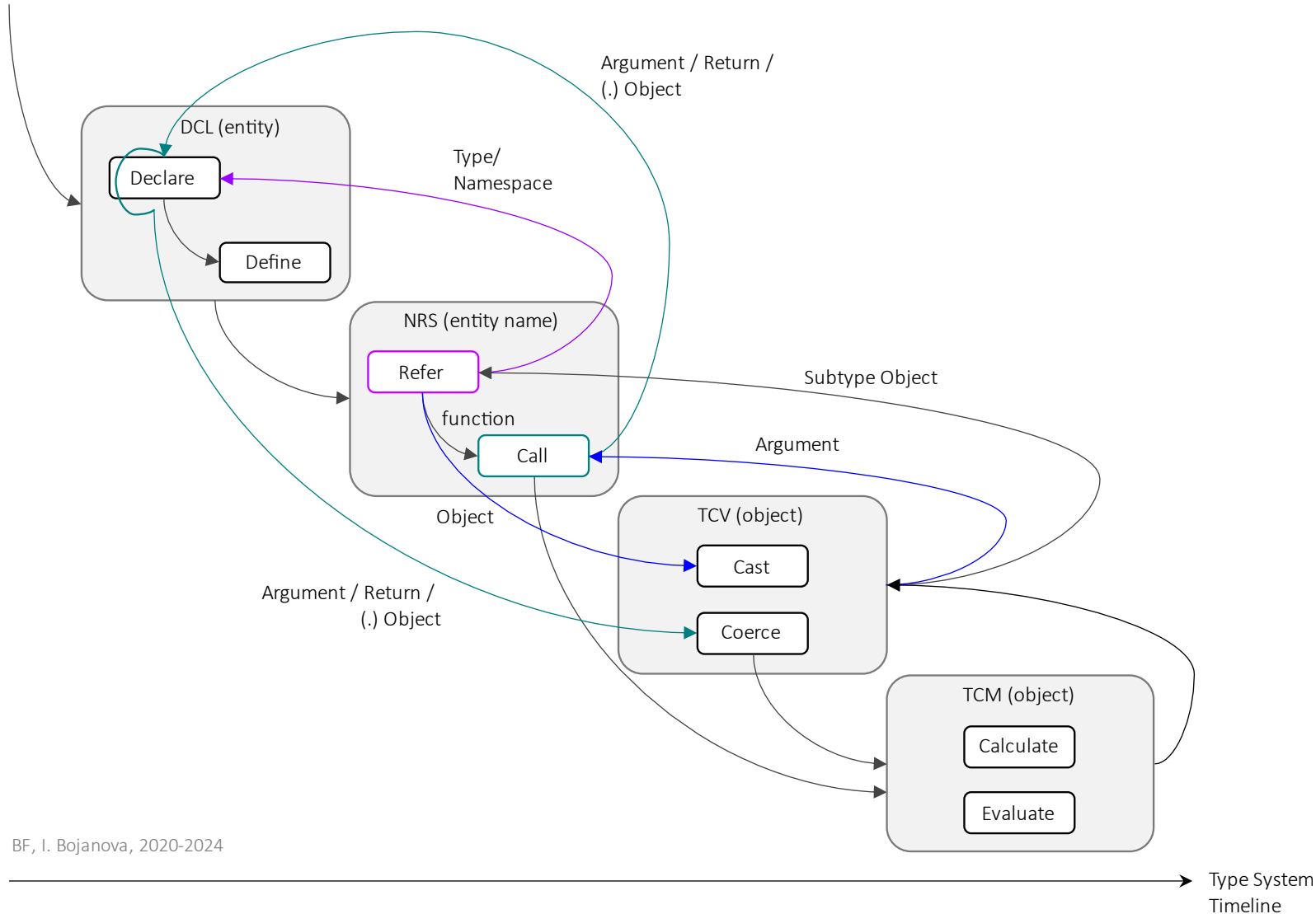
- Identify Secure Code Principles:
  - Data Type Safety

BF Weakness Taxonomies

Bugs Framework (BF)

Memory Corruption/Disclosure (_MEM) Class Type
Memory Use (MUS) Class

Causes

Code Bug
- Missing Code
- Erroneous Code

Data Fault
- Forbidden Address
- Wrong Size

Type Fault
- Casted Pointer

Address Fault
- NULL Pointer
- Wild Pointer
- Dangling Pointer
- Untrusted Pointer
- Over Bounds Pointer
- Under Bounds Pointer
- Wrong Position Pointer

Size Fault
- Not Enough Memory

Operations

Initialize Object
Dereference
Read
Write
Clear

Operands

Data
Type
Address
Size

Consequences

Data Error
- Uninitialized Object

Memory Corruption/Disclosure Final Error
- Not Cleared Object
- NULL Pointer Dereference
- Untrusted Pointer Dereference
- Object Corruption
- Type Confusion
- Use After Deallocate
- Buffer Overflow
- Buffer Underflow
- Buffer Over-Read
- Buffer Under-Read
- Uninitialized Pointer Dereference

Bugs Framework (BF), I. Bojanova, 2014-2024

BF Memory Use (MUS) Class

An object is initialized, read, written, or cleared improperly.

$(Bug, Operation_1, Error_1)$ ← lookup_weakness_triple()

$(Fault_1, Operation_2, Error_2)$ ← lookup_weakness_triple()

$(Fault_{n-1}, Operation_n, Final\ Error)$ ← lookup_weakness_triple()

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

Attributes

Address Kind	Address State	Size Kind	Mechanism	Source Code	Execution Space
Huge	Stack	Actual	Direct	Codebase	Userland
Moderate	Heap	Used	Sequential	Third-Party	Kernel
Little	/other/			Standard Library	Bare-Metal
				Compiler/Interpreter	

Bug | Fault/Error | Final Error | Operation/Operand

BF Data Validation (DVL) Class – Example from _INP Class Type

NIST

Bugs Framework (BF)

Input/Output Check (_INP) Class Type
Data Validation (DVL) Class

Causes	Operations	Consequences
Code Bug	**Validate**	**Data Error**
Missing Code	Sanitize	Invalid Data
Erroneous Code		
	Operands	**Injection Final Error**
Specification Bug	**Data**	Query Injection
Under-Restrictive Policy		Command Injection
Over-Restrictive Policy		Source Code Injection
		Parameter Injection
Data Fault		File Injection
Corrupted Data		
Tampered Data		
Corrupted Policy Data		
Tampered Policy Data		

Bugs Framework (BF), I. Bojanova, 2014-2024

Attributes

Data State	Mechanism	Source Code	Execution Space
Entered	Safelist	Codebase	Local
Stored	Denylist	Third-Party	Admin
In Use	Format	Standard Library	Bare-Metal
Transferred	Length	Compiler/Interpreter	

▢ Bug	▢ Fault/Error	▢ Final Error	☐ Operation/Operand

BF Data Validation (DVL) Class

Data are validated (syntax check) or sanitized (escape, filter, repair) improperly.

$(\text{Bug}, \text{Operation}_1, \text{Error}_1)$ ← lookup_weakness_triple()

$(\text{Fault}_1, \text{Operation}_2, \text{Error}_2)$ ← lookup_weakness_triple()

$(\text{Fault}_{n-1}, \text{Operation}_n, \text{Final Error})$ ← lookup_weakness_triple()

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

I. Bojanova, 2024

NIST
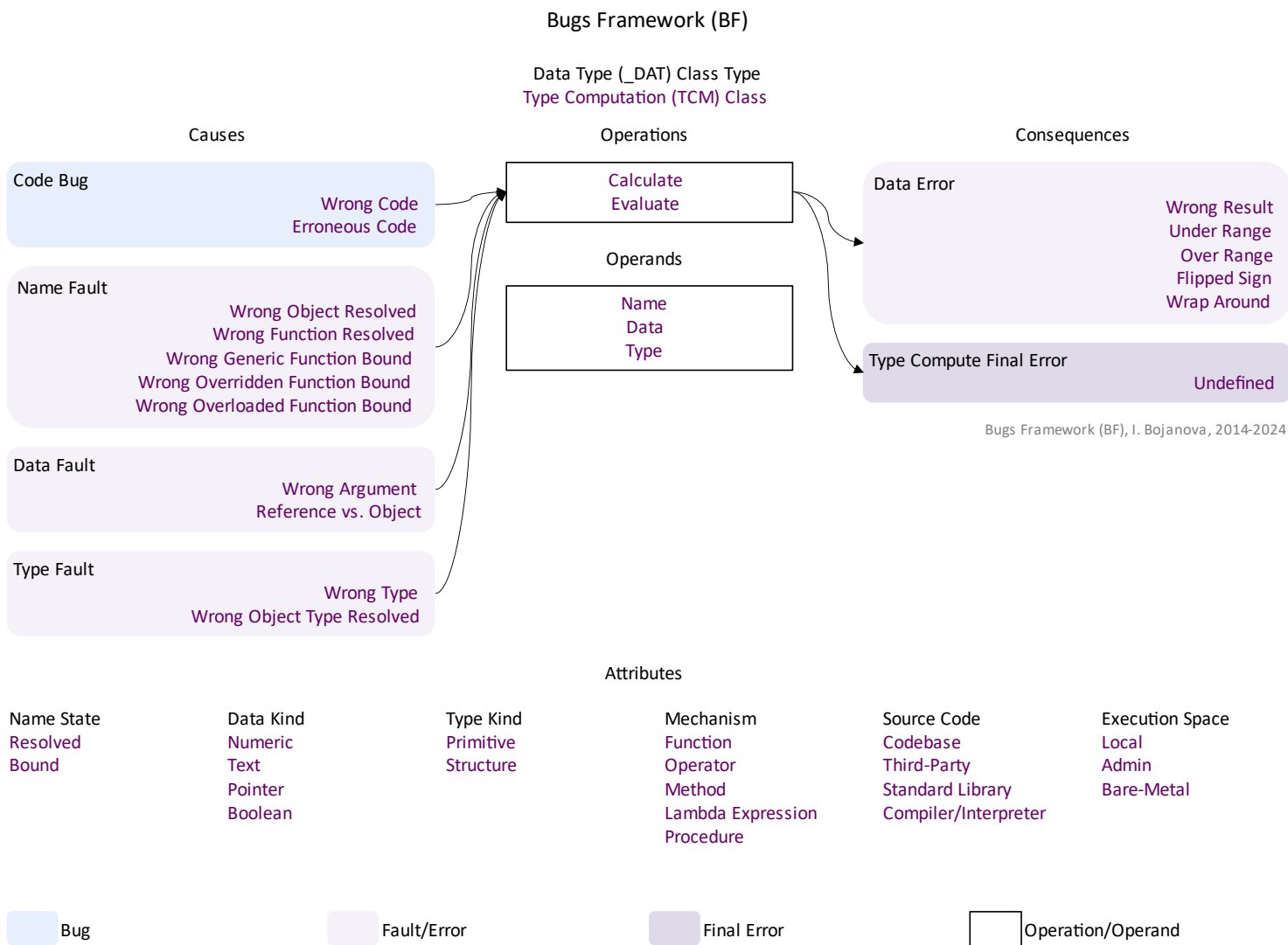
Type Computation (TCM) Class

An arithmetic expression (over numbers, strings, or pointers) is calculated improperly, or a boolean condition is evaluated improperly.

Bugs Framework (BF)

Data Type (_DAT) Class Type
Type Computation (TCM) Class

Causes

Code Bug
Wrong Code
Erroneous Code

Name Fault
Wrong Object Resolved
Wrong Function Resolved
Wrong Generic Function Bound
Wrong Overridden Function Bound
Wrong Overloaded Function Bound

Data Fault
Wrong Argument
Reference vs. Object

Type Fault
Wrong Type
Wrong Object Type Resolved

Operations

Calculate
Evaluate

Operands

Name
Data
Type

Consequences

Data Error
Wrong Result
Under Range
Over Range
Flipped Sign
Wrap Around

Type Compute Final Error
Undefined

Bugs Framework (BF), I. Bojanova, 2014-2024

$(Bug, Operation_1, Error_1)$ ← lookup_weakness_triple()

$(Fault_1, Operation_2, Error_2)$ ← lookup_weakness_triple()

$(Fault_{n-1}, Operation_n, Final Error)$ ← lookup_weakness_triple()

Attributes

Name State	Data Kind	Type Kind	Mechanism	Source Code	Execution Space
Resolved	Numeric	Primitive	Function	Codebase	Local
Bound	Text	Structure	Operator	Third-Party	Admin
	Pointer		Method	Standard Library	Bare-Metal
	Boolean		Lambda Expression	Compiler/Interpreter	
			Procedure		

Bug

Fault/Error

Final Error

Operation/Operand

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

BF Weakness Taxonomies

➢ Structured

   `(bug/fault, operation, error/final error)`

➢ Complete

  no gaps in coverage

➢ Orthogonal

  no overlaps

➢ Language and domain independent

  context-free

➢ Causation rules

  cause-consequence transition by operation

BF Taxonomy – BF.xml

BF Vulnerability Models

BF Vulnerability State Model

BF Vulnerability State Model

Improper State: An `operation, operand₁, …, operandₙ)` tuple with at least one improper element (depicted in purple)

○ Initial State – a weakness caused by a Bug in the Operation, resulting in an Error

○ Propagation State – a weakness caused by a Fault of an Operand, resulting in an Error

○ Final State – a weakness caused by a Fault in an Operand, resulting in a Final Error

△ Failure – a violation of a security requirement caused by an exploit; may result in a Fault starting a new faultsonly vulnerability

↝ Chaining weaknesses underlying a vulnerability

⊕ Converging vulnerabilities

↪ Chaining vulnerabilities

BF, I. Bojanova, 2014-2024

- The bug in at least one of the chains must be fixed to resolve the vulnerability
- Fixing a fault may only mitigate the vulnerability

BF Vulnerability Specification Model

- Chain of (cause, operation, consequence) weakness triples

- Bug   =   improper operation

- Fault   =   improper operand

- Bug Types: Code, Specification

- Fault Types: Name, Data, Type, Address, Size

- Causation within a weakness
- Causation between weaknesses
- Causation between vulnerabilities

- Propagation between weaknesses
- Propagation between vulnerabilities

BF Vulnerability Specification Model

Weakness Type $_1$
Weakness $_1$

Bug Type

| Bug |

Buggy Operation

Operation $_1$
Operand $_{11}$, ... Operand $_{1i}$, ...

Error Type $_1$

Error $_1$

Operand $_{11}$ Kind		Operand $_{15}$ Kind	Mechanism	Source Code	Execution Space
Value $_{11s}$	...	Value $_{15s}$	Value $_{1m}$	Value $_{1c}$	Value $_{1e}$
Operand $_{11}$ State		Operand $_{15}$ State			
Value $_{11k}$		Value $_{15k}$			

(Bug, Operation $_1$, Error $_1$) ← lookup_weakness_triple()

(Operation $_1$, Operation $_2$) ← lookup_operation_flow()

Fault $_1$.Type ← Error $_1$.Type

Weakness Type $_2$
Weakness $_2$

Fault Type $_1$

| Fault $_1$ |

Operation with Faulty Operand

Operation $_2$
Operand $_{21}$, ..., Operand $_{2j}$, ...

Error Type $_2$

Error $_2$

Operand $_{21}$ Kind		Operand $_{25}$ Kind	Mechanism	Source Code	Execution Space
Value $_{21s}$	...	Value $_{25s}$	Value $_{2m}$	Value $_{2c}$	Value $_{2e}$
Operand $_{21}$ State		Operand $_{25}$ State			
Value $_{21k}$		Value $_{25k}$			

(Fault $_1$, Operation $_2$, Error $_2$) ← lookup_weakness_triple()

(Operation $_2$, Operation $_3$) ← lookup_operation_flow()

Fault $_2$.Type ← Error $_2$.Type

...

Weakness Type $_N$
Weakness $_N$

Fault Type $_{N-1}$

| Fault $_{N-1}$ |

Operation with Faulty Operand

Operation $_N$
Operand $_{N1}$, ..., Operand $_{NP}$, ...

Final Error Type

Final Error

Operand $_{N1}$ Kind		Operand $_{N5}$ Kind	Mechanism	Source Code	Execution Space
Value $_{N1s}$	...	Value $_{N5s}$	Value $_{Nm}$	Value $_{Nc}$	Value $_{Ne}$
Operand $_{N1}$ State		Operand $_{N5}$ State			
Value $_{N1k}$		Value $_{N5k}$			

(Fault $_{n-1}$, Operation $_n$, Final Error)   ← lookup_weakness_triple()
(Final Error, Exploit $_1$)   ← flow_lookup()
...
(Final Error, Exploit $_F$)   ← flow_lookup()

Bug Type – Code or Specification
Fault/Error Type – Data, Name, Type, Address, or Size
→ Causation within weaknesses – by valid (cause, operation, consequence) triples
↪ Causation between weaknesses – by flow of operations
Propagation between weaknesses – by same error/fault value or type

Bug   Operation

Fault / Error   Final Error   Failure

Failure Type $_1$
Failure $_1$

Failure Type $_F$
Failure $_F$

Exploit $_1$
Vector $_1$

...

Exploit $_F$
Vector $_F$

BF Formal Language

BF Context Free Grammar (CFG)

$$G = (V, \Sigma, R, S) \qquad\qquad (1)$$

, where:

- Σ defines the BF lexis (the alphabet of the CFG) as a finite set of tokens (terminals) comprised by the sets of BF taxons and BF symbols (see Listing 3)

$$\Sigma = \{\alpha \mid \alpha \in \Sigma BF\,taxons \cup \Sigma BF\,symbols\}$$

- V and R define the BF syntax as
  - a finite set of variables (nonterminals)

$$V = \{S, V_1, \ldots, V_n\}$$

  - and a finite set of syntactic rules (productions) in the form

$$R = \{A \longmapsto \omega \mid A \in V \wedge \omega \in (V \cup \Sigma)^*\}$$

  , where:

  $(V \cup \Sigma)^*$ is a string of tokens and/or variables

  $A \longmapsto \omega$ means any variable A occurrence may be replaced by ω.

- $S \in V$ is the predefined start variable from which all BF specifications derive.

BF Formal Language

The formal language is defined as the set of all strings of tokens ω derivable from the start variable S.

$$L(G) = \{\omega \in \Sigma^* : S \overset{*}{\Longrightarrow} \omega\} \qquad (2)$$

, where:

- Σ^* is the set of all possible strings that can be generated from Σ tokens

- S is the start variable

- $\alpha \overset{*}{\Longrightarrow} \beta$ means string α derives string β

Note that ω must be in Σ^*, the set of strings made from terminals. Strings involving non-terminals are not part of the language.

BF CFG Lexis

$$\Sigma = \{\Sigma BFtaxons, \Sigma BFsymbols\} \qquad (3)$$

, where

$$\Sigma BFtaxons = \{\Sigma Category, \Sigma ClassType, \Sigma Class, \Sigma BugType, \Sigma Bug,$$
$$\Sigma Operation, \Sigma OperationAttributeType,$$
$$\Sigma FaultType, \Sigma Fault, \Sigma OperandAttributeType, \Sigma OperandAttribute,$$
$$\Sigma FinalErrorType, \Sigma FinalError\}$$

$$\Sigma BFsymbols = \{\ \rightarrow, \hookrightarrow, \oplus\}$$

$$\Sigma Category = \{'Weakness', 'Failure'\}$$
$$\Sigma ClassType = \{'\_INP', '\_DAT', '\_MEM', ...\}$$
$$\Sigma Class = \{'DVL', 'DVR', 'DCL', 'NRS', 'TCV', 'TCM', 'MAD', 'MMN', 'MUS', ...\}$$

$$\Sigma Operation = \{'Validate', 'Sanitize', 'Verify', 'Correct', 'Declare', 'Define', 'Refer',$$
$$'Call', 'Cast', 'Coerce', 'Calculate', 'Evaluate', 'InitializePointer',$$
$$'Reposition', 'Reassign', 'Allocate', 'Extend', 'Reallocate - Extend',$$
$$'Deallocate', 'Reduce', 'Reallocate - Reduce', 'InitializeObject',$$
$$'Dereference', 'Read', 'Write', 'Clear', 'Generate/Select', 'Store',$$
$$'Distribute', 'Use' ...\}$$

$$\Sigma Operation = \{'Validate', 'Sanitize', 'Verify', 'Correct', 'Declare', 'Define', 'Refer',$$
$$'Call', 'Cast', 'Coerce', 'Calculate', 'Evaluate', 'InitializePointer',$$
$$'Reposition', 'Reassign', 'Allocate', 'Extend', 'Reallocate - Extend',$$
$$'Deallocate', 'Reduce', 'Reallocate - Reduce', 'InitializeObject',$$
$$'Dereference', 'Read', 'Write', 'Clear', 'Generate/Select', 'Store',$$
$$'Distribute', 'Use' ...\}$$

$$\Sigma BugType = \{'CodeDefect', 'SpecificationDefect'\}$$
$$\Sigma Bug = \{'MissingCode', 'ErroneousCode', 'Under - RestrictivePolicy',$$
$$'Over - RestrictivePolicy', 'WrongCode', 'MissingModifier',$$
$$'WrongModifier', 'AnonymousScope', 'WrongScope',$$
$$'MissingQualifier', 'WrongQualifier', 'MismatchedOperation', ...\}$$

$$\Sigma FinalErrorType = \{'Injection', 'Access', 'TypeCompute',$$
$$'MemoryCorruption/Disclosure', ...\}$$

$$\Sigma FinalError = \{'QueryInjection', 'CommandInjection', 'SourceCodeInjection',$$
$$'ParameterInjection', 'FileInjection', 'WrongAccessObject',$$
$$'WrongAccessType', 'WrongAccessFunction', 'Undefined',$$
$$'MemoryLeak', 'MemoryOverflow', 'DoubleDeallocate',$$
$$'ObjectCorruption', 'NotClearedObject',$$
$$'NULLPointerDereference', 'UntrustedPointerDereference',$$
$$'TypeConfusion', 'UseAfterDeallocate', 'BufferOverflow',$$
$$'BufferUnderflow', 'BufferOver - Read', 'BufferUnder - Read$$

BF CFG Syntax

$$S ::= \big(Vulnerability\, (\oplus Vulnerability)?\, Failure+\big) + \varepsilon \qquad\qquad (4)$$

$$
\begin{aligned}
Vulnerability &::= +Weakness \\
Weakness &::= Cause\, Operation\, Consequence \\
Cause &::= Bug \mid Fault \\
Consequence &::= Error \mid FinalError
\end{aligned}
$$

$$S ::= \big(Vulnerability\, (\oplus Vulnerability)?\, Failure+\big) + \varepsilon \qquad\qquad (5)$$

$$
\begin{aligned}
Vulnerability &::= SingleWeakness \\
&\quad \mid FirstWeakness\, (Weakness+)\, LastWeakness \\
SingleWeakness &::= Bug\, Operation\, FinalError \\
FirstWeakness &::= Bug\, Operation\, (Error \mid FinalError) \\
Weakness &::= Fault\, Operation\, Error \\
LastWeakness &::= Error\, Operation\, FinalError
\end{aligned}
$$

BF Syntax – LL(1) Grammar

$$S ::= \textit{Vulnerability Converge\_Failure} \qquad\qquad (6)$$

$$\textit{Vulnerability} ::= \textit{Bug\_Fault Operation OperAttrs\_Error\_FinalError}$$

$$\textit{Bug\_Fault} ::= \textit{Bug}$$
$$| \textit{Fault}$$

$$\textit{OperAttrs\_Error\_FinalError} ::= \textit{OperationAttribute OperAttrs\_Error\_FinalError}$$
$$| \textit{Error Fault OprndAttrs\_Operation}$$
$$| \textit{FinalError}$$

$$\textit{OprndAttrs\_Operation} ::= \textit{OperandAttribute OprndAttrs\_Operation}$$
$$| \textit{Operation OperAttrs\_Error\_FinalError}$$

$$\textit{Converge\_Failure} ::= \oplus \textit{ Vulnerability Converge\_Failure}$$
$$| \textit{Vector Exploit NextVulner\_Failure}$$

$$\textit{NextVulner\_Failure} ::= \textit{Fault OprndAttrs\_Operation}$$
$$| \textit{Failure } \varepsilon$$

$$SyntaxRules:$$

$$(7)$$

$$S ::= Vulnerability\ Converge\_Failure$$

$$Vulnerability ::= Bug\_Fault\ Operation\ OperAttrs\_Error\_FinalError$$

$$Bug\_Fault ::= Bug$$
$$\mid Fault$$

$$OperAttrs\_Error\_FinalError ::= OperationAttribute\ OperAttrs\_Error\_FinalError$$
$$\mid Error\ Fault_1\ OprndAttrs\_Operation$$
$$\mid FinalError$$

$$OprndAttrs\_Operation ::= OperandAttribute\ OprndAttrs\_Operation$$
$$\mid Operation_k\ OperAttrs\_Error\_FinalError$$

$$Converge\_Failure ::= \oplus\ Vulnerability\ Converge\_Failure$$
$$\mid Vector\ Exploit\ NextVulner\_Failure$$

$$NextVulner\_Failure ::= Fault_2\ OprndAttrs\_Operation$$
$$\mid Failure\ \varepsilon$$

$$SemanticRules:$$

$$(Bug,\ Operation_1,\ Error) \leftarrow lookup\_weakness\_triple()$$
$$(Bug,\ Operation_1,\ FinalError) \leftarrow lookup\_weakness\_triple()$$
$$(Fault_1,\ Operation_k,\ Error),\ k>1 \leftarrow lookup\_weakness\_triple()$$
$$(Fault_1,\ Operation_k,\ FinalError),\ k>1 \leftarrow lookup\_weakness\_triple()$$
$$(Operation_1,...Operation_k),\ k>1 \leftarrow lookup\_operation\_flow()$$
$$Fault_1 \leftarrow if\ (Fault_1.ClassType == Error.ClassType)\ then\ Error$$

$$Predicates:$$

$$Fault_1.Type == Error.Type$$
$$Vector.Type == FinalError.Type$$
$$Fault_2.Type == ExploitResult.Type$$

BF Specifications of CWEs

BFCWE Dataset

```xml
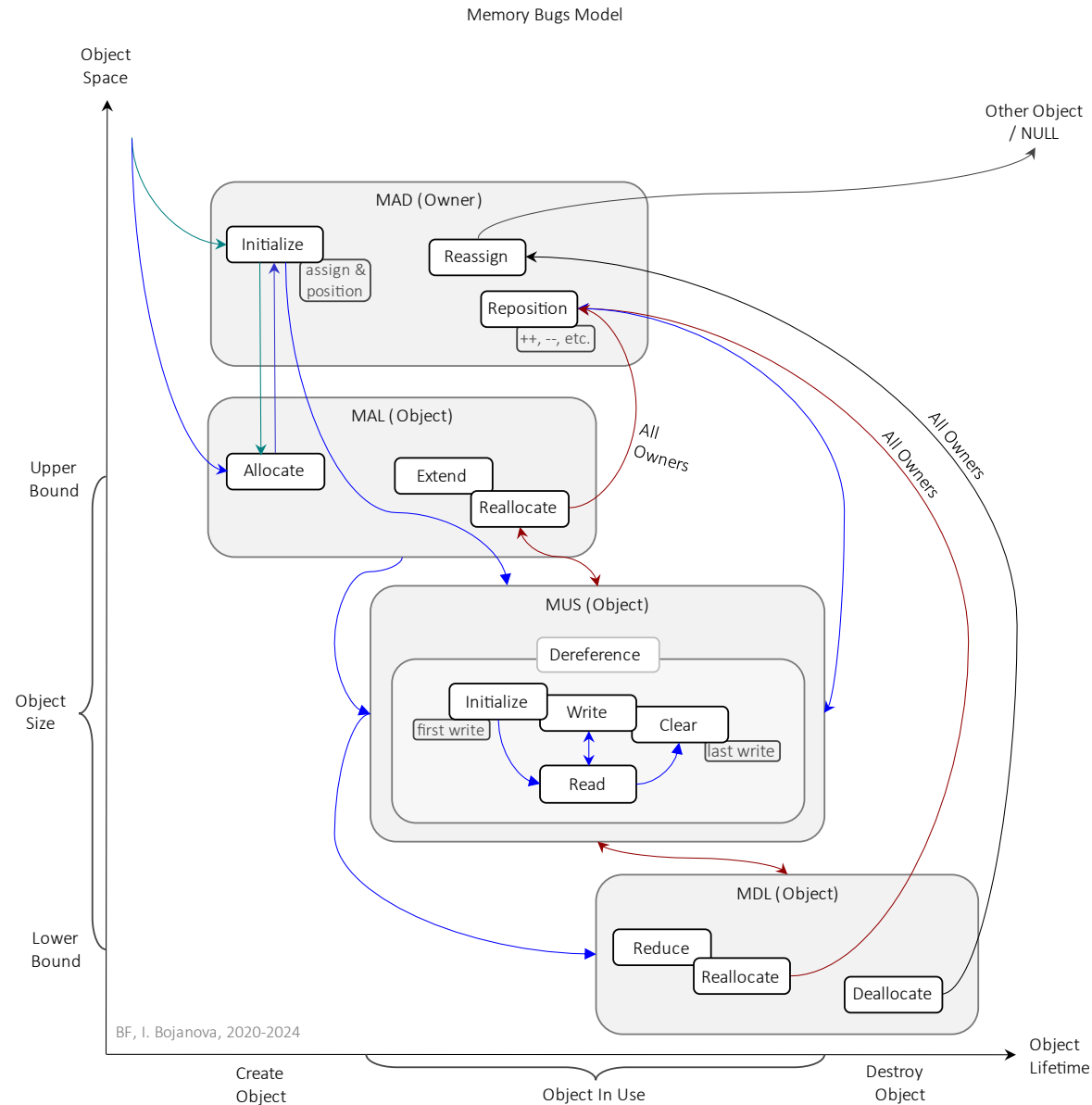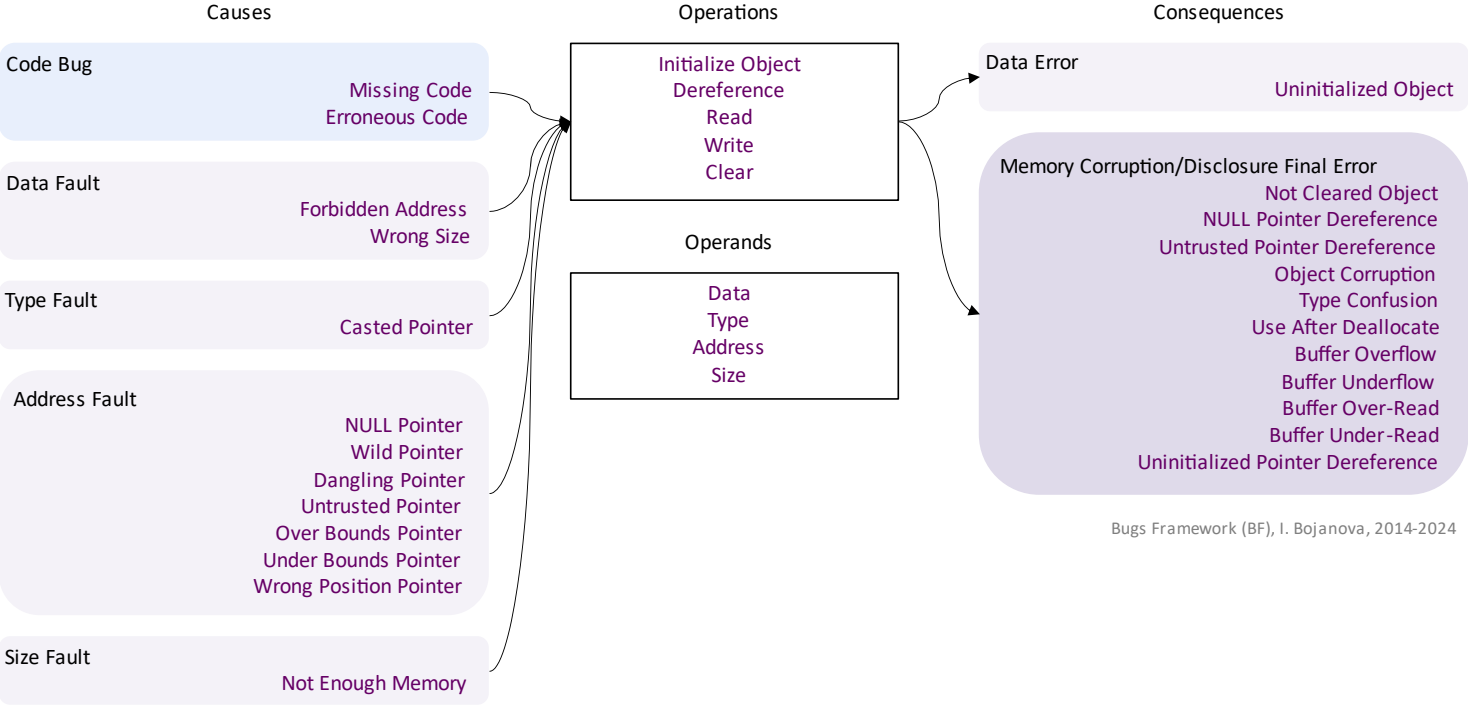<?xml version="1.0" encoding="utf-8"?>
<!--Bugs Framework (BF), BFCWE Tool, I. Bojanova, NIST, 2020-2024-->
<BFCWE-Dataset>
  <CWE ID="20">
    <BFCWE Cause="Missing Code" Operation="Verify" Consequence="Wrong Value" />
    <BFCWE Cause="Missing Code" Operation="Verify" Consequence="Incosnistent Value" />
    <BFCWE Cause="Missing Code" Operation="Verify" Consequence="Wrong Type" />
    <BFCWE Cause="Erroneous Code" Operation="Verify" Consequence="Wrong Value" />
    <BFCWE Cause="Erroneous Code" Operation="Verify" Consequence="Incosnistent Value" />
    <BFCWE Cause="Erroneous Code" Operation="Verify" Consequence="Wrong Type" />
    <BFCWE Cause="Missing Code" Operation="Validate" Consequence="Invalid Data" />
    <BFCWE Cause="Erroneous Code" Operation="Validate" Consequence="Invalid Data" />
  </CWE>
  <CWE ID="22">
    <BFCWE Cause="Under-Restrictive Policy" Operation="Sanitize" Consequence="File Injection" />
  </CWE>
  <CWE ID="23">
    <BFCWE Cause="Under-Restrictive Policy" Operation="Sanitize" Consequence="File Injection" />
  </CWE>
  <CWE ID="24">
    <BFCWE Cause="Under-Restrictive Policy" Operation="Sanitize" Consequence="File Injection" />
  </CWE>
  <CWE ID="25">
    <BFCWE Cause="Under-Restrictive Policy" Operation="Sanitize" Consequence="File Injection" />
  </CWE>
  <CWE ID="26">
    <BFCWE Cause="Under-Restrictive Policy" Operation="Sanitize" Consequence="File Injection" />
  </CWE>
  <CWE ID="27">
    <BFCWE Cause="Under-Restrictive Policy" Operation="Sanitize" Consequence="File Injection" />
  </CWE>
  <CWE ID="28">
    <BFCWE Cause="Under-Restrictive Policy" Operation="Sanitize" Consequence="File Injection" />
  </CWE>
  <CWE ID="29">
    <BFCWE Cause="Under-Restrictive Policy" Operation="Sanitize" Consequence="File Injection" />
  </CWE>
  <CWE ID="30">
    <BFCWE Cause="Under-Restrictive Policy" Operation="Sanitize" Consequence="File Injection" />
  </CWE>
  <CWE ID="31">

  <CWE ID="125">
    <BFCWE Cause="Over Bounds Pointer" Operation="Read" Consequence="Buffer Over-Read" />
    <BFCWE Cause="Under Bounds Pointer" Operation="Read" Consequence="Buffer Under-Read" />
  </CWE>
  <CWE ID="126">
    <BFCWE Cause="Over Bounds Pointer" Operation="Read" Consequence="Buffer Over-Read" />
  </CWE>
  <CWE ID="127">
    <BFCWE Cause="Under Bounds Pointer" Operation="Read" Consequence="Buffer Under-Read" />
  </CWE>
  <CWE ID="128">
    <BFCWE Operation="Calculate" Consequence="Wrap Around" />
  </CWE>
  <CWE ID="129">
    <BFCWE Cause="Missing Code" Operation="Verify" Consequence="Wrong Value" />
    <BFCWE Cause="Erroneous Code" Operation="Verify" Consequence="Wrong Value" />
  </CWE>
  <CWE ID="130">
    <BFCWE Cause="Missing Code" Operation="Verify" Consequence="Inconsistent Value" />
    <BFCWE Cause="Erroneous Code" Operation="Verify" Consequence="Inconsistent Value" />
  </CWE>
  <CWE ID="131">
    <BFCWE Operation="Calculate" />
  </CWE>
  <CWE ID="134">
    <BFCWE Cause="Missing Code" Operation="Validate" Consequence="Parameter Injection" />
  </CWE>
  <CWE ID="135">
    <BFCWE Operation="Calculate" Consequence="Wrong Result" />
  </CWE>
  <CWE ID="138">
    <BFCWE Cause="Missing Code" Operation="Sanitize" Consequence="Invalid Data" />
    <BFCWE Cause="Erroneous Code" Operation="Sanitize" Consequence="Invalid Data" />
  </CWE>
  <CWE ID="140">
    <BFCWE Cause="Missing Code" Operation="Sanitize" Consequence="Parameter Injection" />
    <BFCWE Cause="Erroneous Code" Operation="Sanitize" Consequence="Parameter Injection" />
  </CWE>
```

CWE-125 – Two BF Specifications

NIST

BF Specifications of CWE-125

_MEM Weakness	Address Fault	Operation	Memory Corruption/Disclosure Final Error
Memory Use (MUS)	Over Bounds Pointer	Read (Improper Operand: Address)	Buffer Over-Read

_MEM Weakness	Address Fault	Operation	Memory Corruption/Disclosure Final Error
Memory Use (MUS)	Under Bounds Pointer	Read (Improper Operand: Address)	Buffer Under-Read
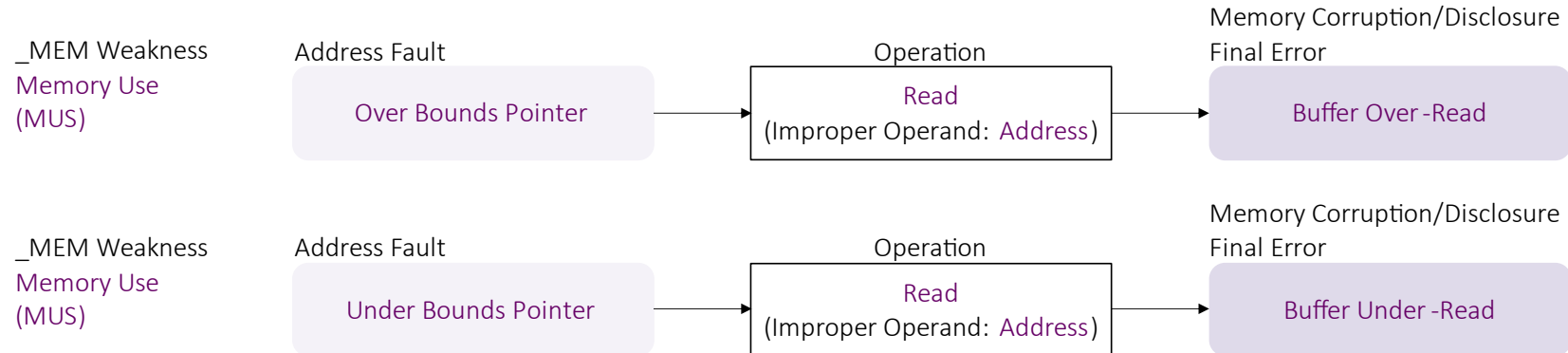
BF, I. Bojanova, 2014-2024

Class Type	Definition
Memory Corruption/Disclosure (_MEM)	Bugs/weaknesses allowing a memory corruption/disclosure exploit.
Class	**Definition**
Memory Use (MUS)	An object is initialized, read, written, or cleared improperly.
Operation	**Definition**
Read	Use the value of an object's data.
Cause	**Definition**
Address Fault	The object address in use is wrong.
Over Bounds Pointer	Holds an address above the upper boundary of its object.
Under Bounds Pointer	Holds an address above the lower boundary of its object.
Consequence	**Definition**
Memory Corruption/Disclosure Final Error	An exploitable or undefined system behavior caused by memory addressing, allocation, use, and deallocation bugs.
Buffer Over-Read	Reading above the upper bound of an object.
Buffer Under-Read	Reading below the lower bound of an object.

Data Type CWEs by BF Operation

- Data Type CWEs
  (incl. Integer Overflow, Juggling, and Pointer Arithmetics) –
  mapped by BF DCL, RNS, TCV, TCM operation

Pie chart values: 16.7, 33.3, 9, 2.6, 1.3, 6.4, 19.2, 14.1

CWEs by DTC, NRS, TCV, and TCM operation:

- DCL Declare
- DCL Define
- NRS Refer
- NRS Call
- TCV Cast
- TCV Coerce
- TCM Calculate
- TCM Evaluate

CWEs by Abstraction:

- Pillar
- Class
- Base
- Variant
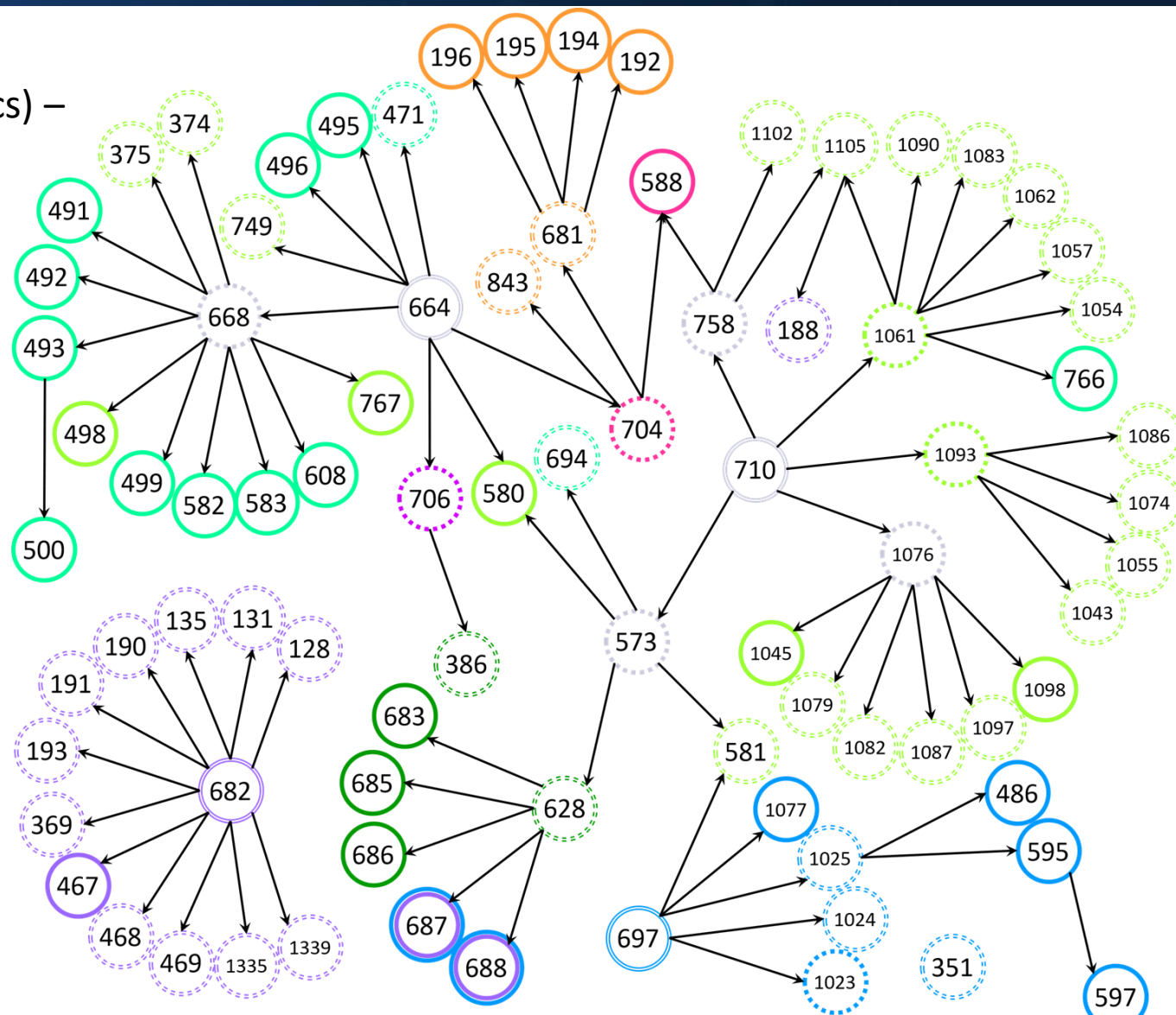
BF Specifications of CVEs

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.

https://nvd.nist.gov/vuln/detail/CVE-2014-0160

Weakness Enumeration

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

Heartbleed (CVE-2014-0160)

NIST

[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.

```
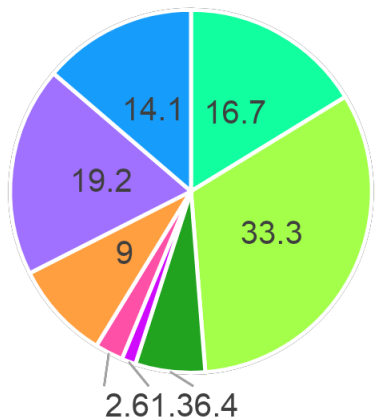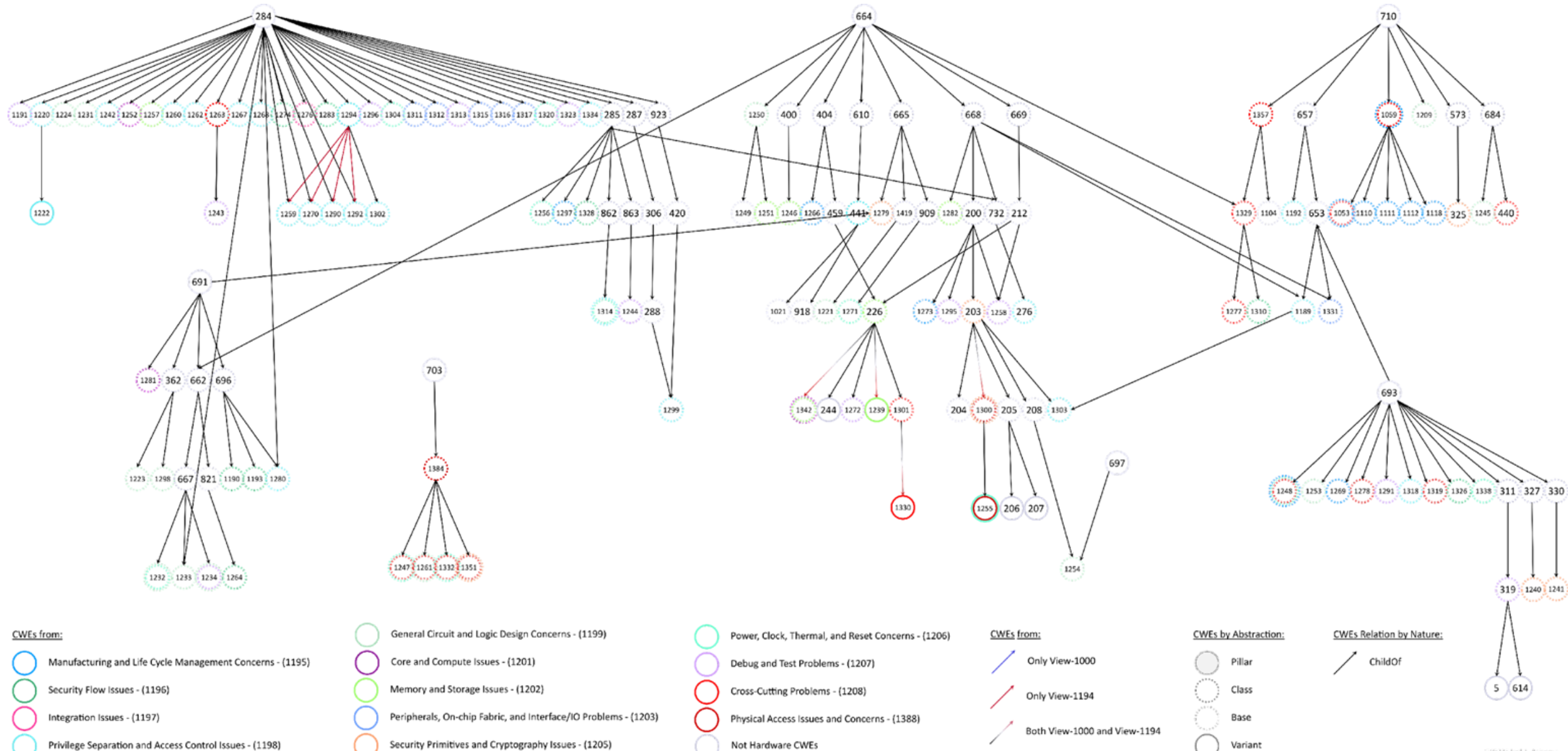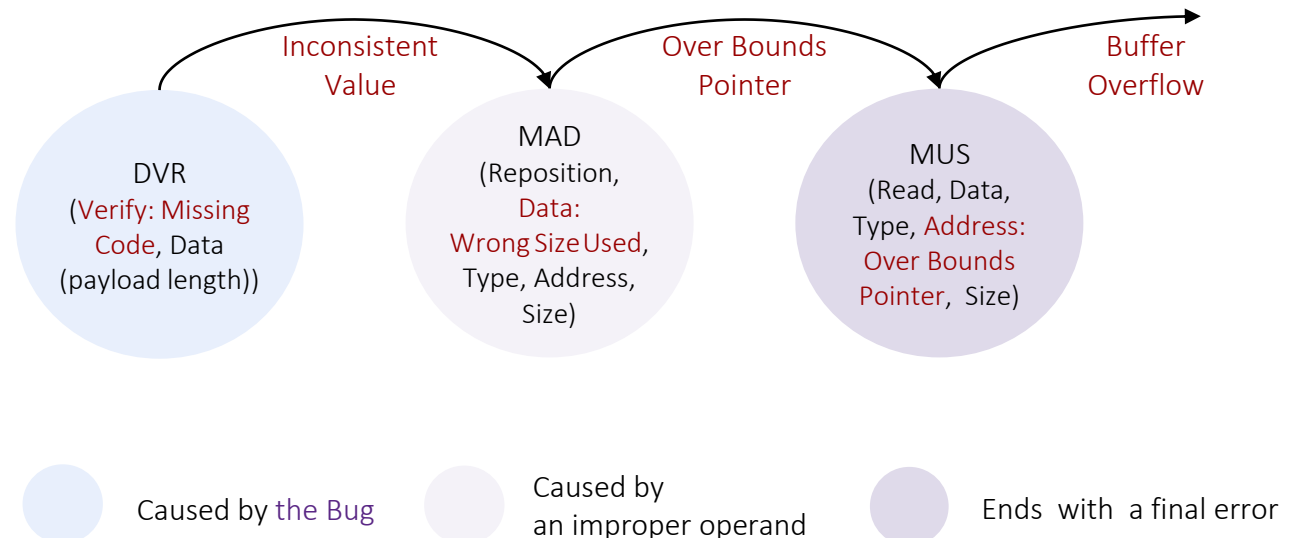1448 dtls1_process_heartbeat(SSL *s)
1449    {unsigned char *p = &s->s3->rrec.data[0], *pl;
1451    unsigned short hbtype;
1452    unsigned
1450    int payload;
1453    unsigned int padding = 16; /* Use minimum padding */
1454
1455    /* Read type and payload length first */
1456    hbtype = *p++;
1457    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
```

Inconsistent Value → Over Bounds Pointer → Buffer Overflow

DVR
(Verify: Missing Code, Data
(payload length))

MAD
(Reposition,
Data:
Wrong Size Used,
Type, Address,
Size)

MUS
(Read, Data,
Type, Address:
Over Bounds
Pointer, Size)

Caused by the Bug

Caused by
an improper operand

Ends with a final error

I. Bojanova, 2024

BF States of CVE-2014-0160 (Heartbleed)
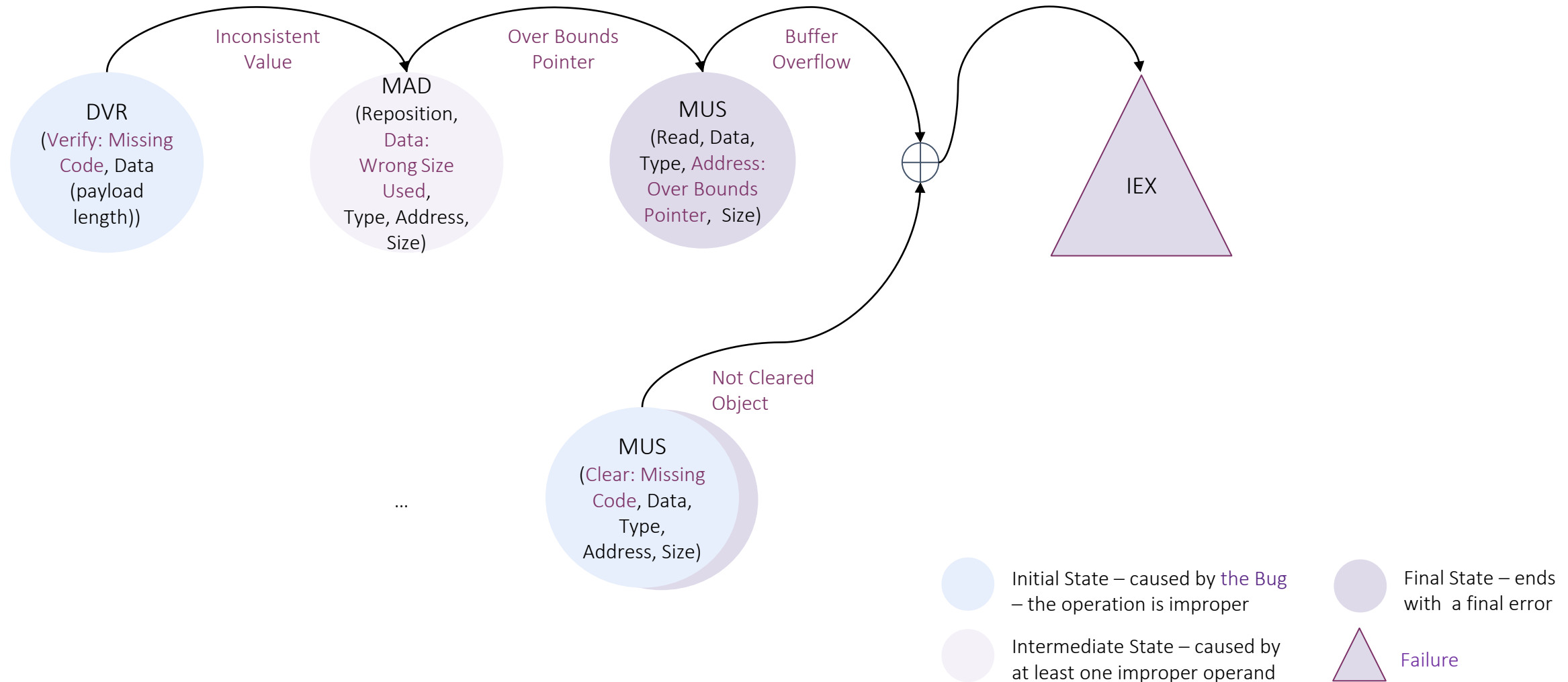
BF Specification of CVE-2014-0160 (Heartbleed)

I. Bojanova, 2024
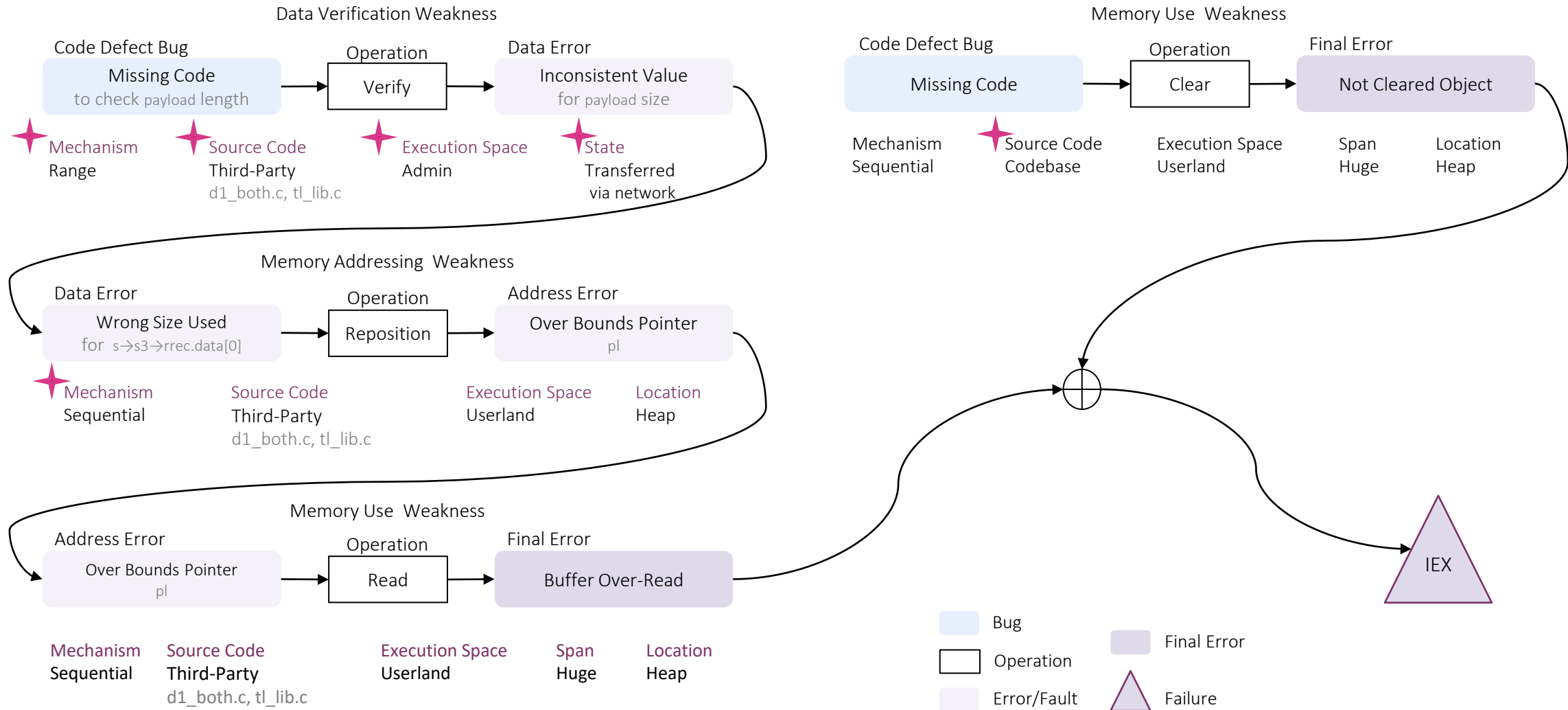
BF Tool – BF Specification of Heartbleed
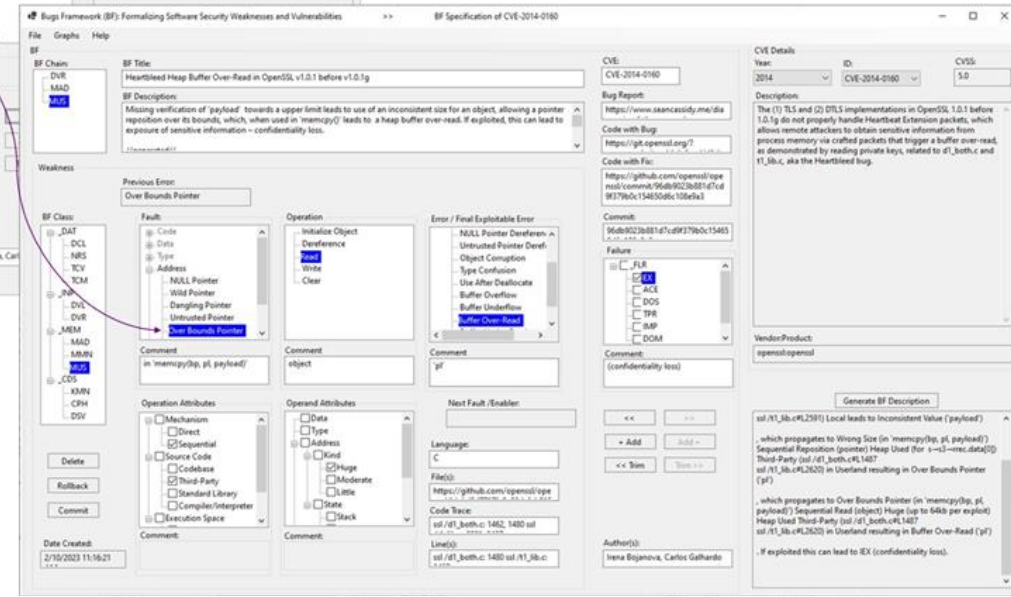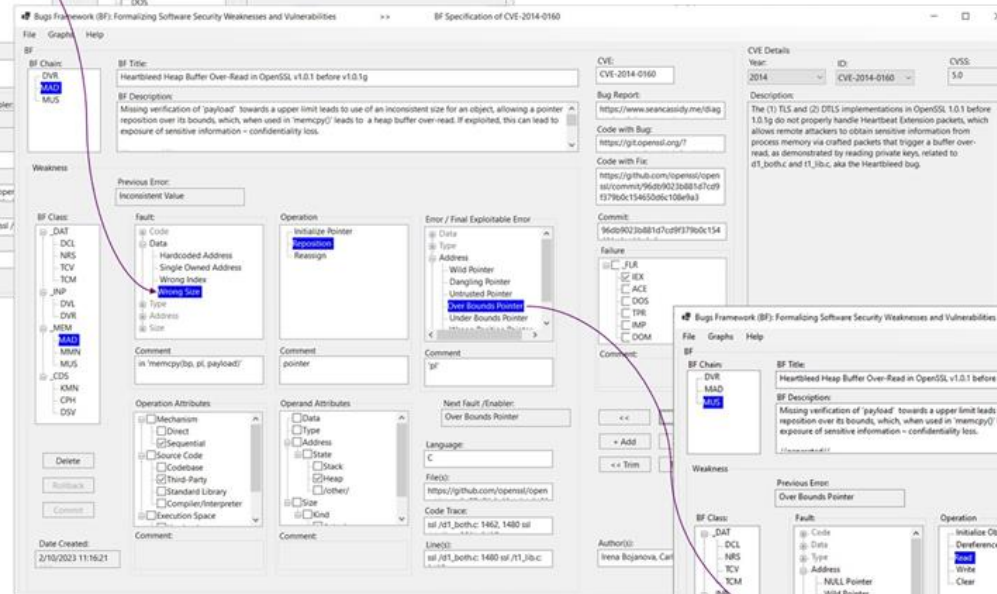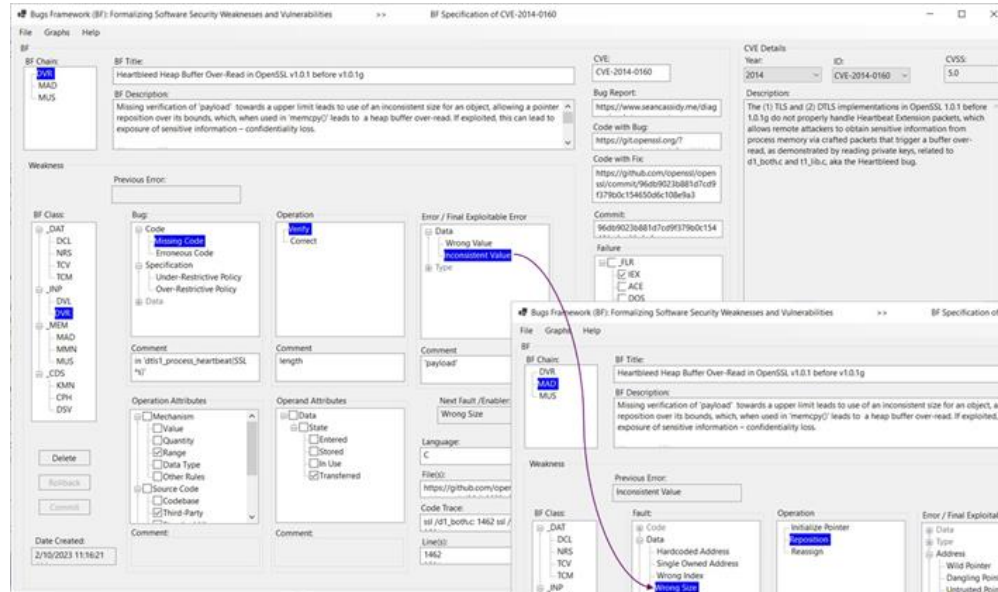
CVE-2014-0160 - Heartbleed.bfcve

```xml
<?xml version="1.0" encoding="utf-8"?>
<CVE Name="1 CVE-2014-0160">
    <BugWeakness Type="_INP" Class="DVR">
        <Cause Type="The Bug">Missing Code</Cause>
        <Operation>Verify</Operation>
        <Consequence Comment="for payload size" Type="Improper Data">Inconsistent Value</Consequence>
        <Attributes>...</Attributes>
    </BugWeakness>
    <Weakness Type="_MEM" Class="MAD">
        <Cause Comment="(for   s→s3→rrec.data[0])" Type="Improper Data">Wrong Size Used</Cause>
        <Operation>Reposition</Operation>
        <Consequence Type="Improper Address">Over Bounds Pointer</Consequence>
        <Attributes>
            <Operation>
                <Attribute Type="Mechanism">Sequential</Attribute>
                <Attribute Comment="d1_both.c and tl_lib.c" Type="Source Code">Codebase</Attribute>
                <Attribute Type="Execution Space">Userland</Attribute>
            </Operation>
            <Operand Name="Object Address">
                <Attribute Type="Location">Heap</Attribute>
            </Operand>
        </Attributes>
    </Weakness>
    <Weakness Type="_MEM" Class="MUS">
        <Cause Comment="(for   s→s3→rrec.data[0])" Type="Improper Address">Over Bounds Pointer</Cause>
        <Operation>Read</Operation>
        <Consequence Type="Memory Error">Buffer Overflow</Consequence>
        <Attributes>...</Attributes>
    </Weakness>
    <Failure Type="_FLR" Class="IEX">
        <Cause Type="Memory Error">Buffer Overflow</Cause>
```

I. Bojanova, 2024

BFCVE Dataset

BF Vulnerability Classification

BF Vulnerability Classification Model

```
]with cweClass as (
select distinct c.Type, class = c.Name, wo.cwe
from bf.class c
inner join bf.operation o on c.Name = o.Class
inner join cwebf.operation wo on o.Name = wo.operation
)
select m.cve [CVE], m.cwe [CWE], n.score [CVSS], ci.url [CodeWithFix], c.Type [BFClassType],
       c.class [BFClass], v.cause [Cause], v.operation [Operation], v.consequence [Consequence]
from cweClass c
inner join nvd.mapCveCwe m on m.cwe = c.cwe
inner join nvd.cve n on m.cve = n.cve
inner join gitHubVul.cve u on u.cve = n.cve
inner join gitHubVul.commitId ci on ci.id = u.commitId
inner join cwe.cwe w on w.id = m.cwe
inner join cwebf.specification s on s.cwe = m.cwe
inner join cwebf.mainWeakness mw on mw.mainWeakness = s.mainWeakness
inner join bf.validWeakness v on v.id = mw.weakness
left outer join cwebf.otherWeakness cw on cw.cwe = m.cwe and cw.mainWeakness = s.mainWeakness
left outer join bf.validWeakness vv on vv.id = cw.weakness
left outer join bf.operation oo on oo.Name = vv.operation
left outer join bf.class cc on oo.Class = cc.Name
where (c.Type = '_MEM')
order by  n.score desc, m.cve, s.cwe, cw.chainId
```

NIST

NVD's One-to-Five Year Plan

Once the NVD is up and running, Brewer said the program will consider new approaches to improving its processes within the next one to five years, especially around software identification.

Some of the ideas include:

▶ **Involving more partners:** Being able to have outside parties submit CPE data for the CPE Dictionary in ways that scale to fit the ever-growing number of IT products

▶ **Software identification improvements:** Dealing with software identification in the NVD in a way that scales with growing complexities (the adoption of PURLS is considered)

▶ **New types of data:** Developing capabilities to publish additional kinds of data to the NVD (e.g. from EPSS, NIST Bugs Framework)

▶ **New use cases:** Developing a way to make NVD data more consumable and more customizable to targeted use cases (e.g. getting email alerts from NVD when CVEs are published)

▶ **CVE JSON 5.0:** Expanding the NVD's capabilities to utilize new data points available in CVE JSON 5.0

▶ **Automation:** Developing a way to automate at least some CVE analysis activities

https://www.infosecurity-magazine.com/news/nist-unveils-new-nvd-consortium/

BF in Security Research

Machine readable formats of:

- BF taxonomy
- BFCWE specifications
- BFCVE specifications
- Vulnerability classifications

✓ Projects related to:
  - ○ Vulnerability specification generation
  - ○ Bug detection
  - ○ Vulnerability analysis and remediation
  - ○ Security failures and risks

BF – Potential Impact

BF – Potential Impacts

- Allow precise communication about security bugs, weaknesses, and vulnerabilities

- ML/AI bug finding, vulnerability analysis, and resolution

- Help identify exploit mitigation techniques.
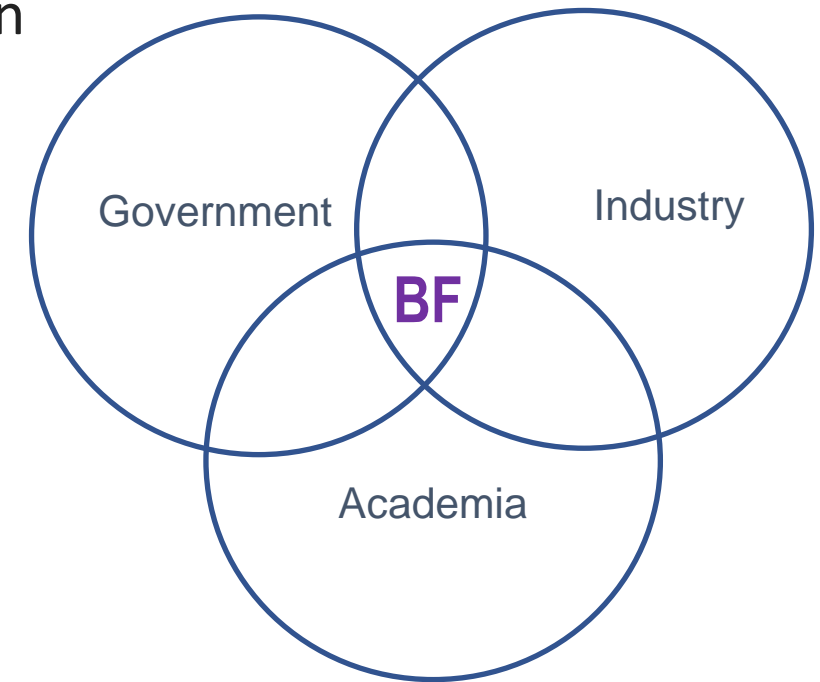
Questions

BF Contact

Irena Bojanova, BF PI & Lead

irena.bojanova@nist.gov

https://samate.nist.gov/BF/
https://usnistgov.github.io/BF/