Bugs Framework (BF)

Formalizing Cybersecurity Weaknesses and Vulnerabilities

National Defense Industrial Association (NDIA)
Trust & Assurance Committee (T&AC)

March 14→ 28, 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

- Imprecise descriptions
- Unclear causality
- Gaps in coverage
- Overlaps in coverage
- Wrong NVD assignments
- No tracking methodology
- No tools

★ Focus is on descriptions

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

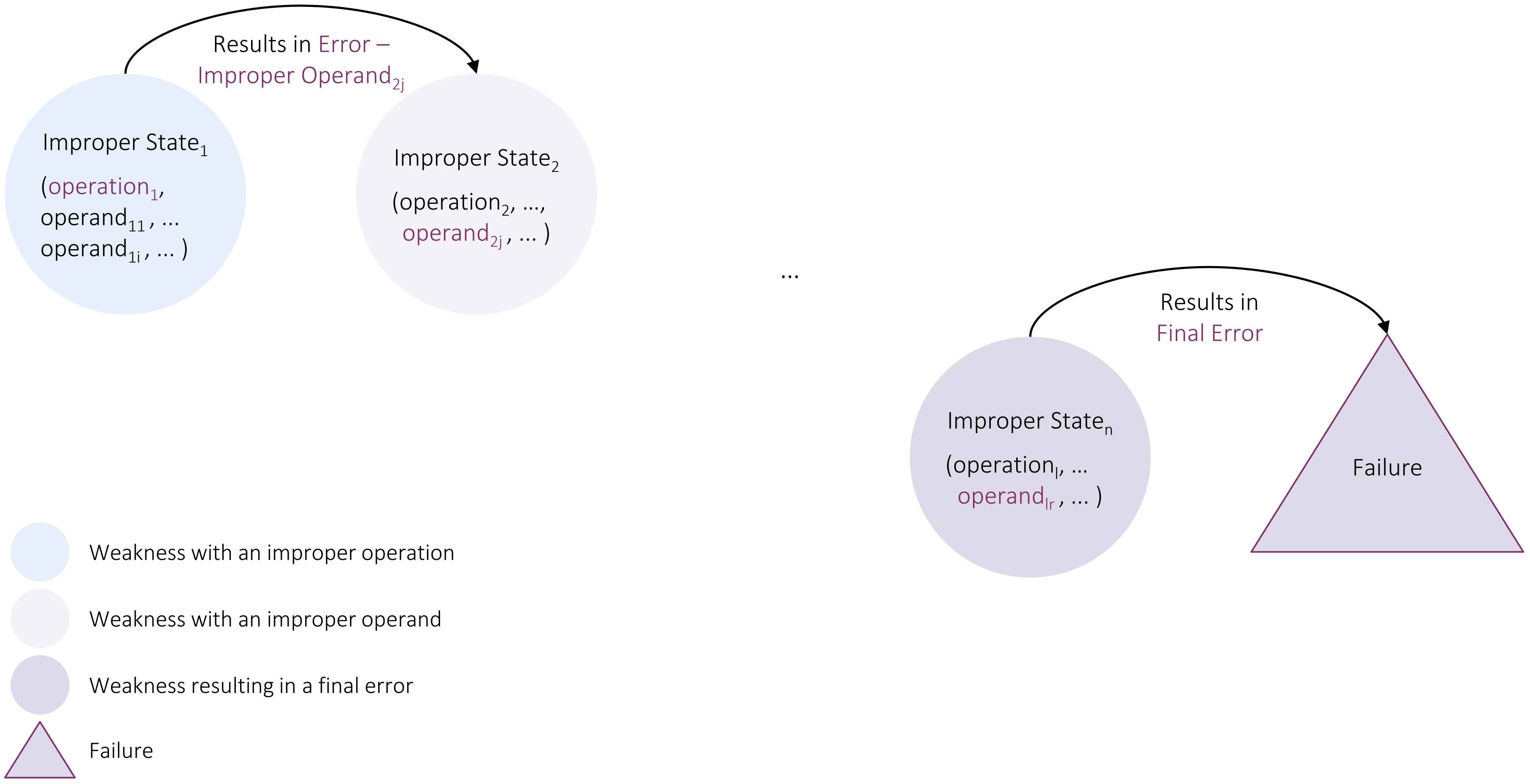
BF Approach

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

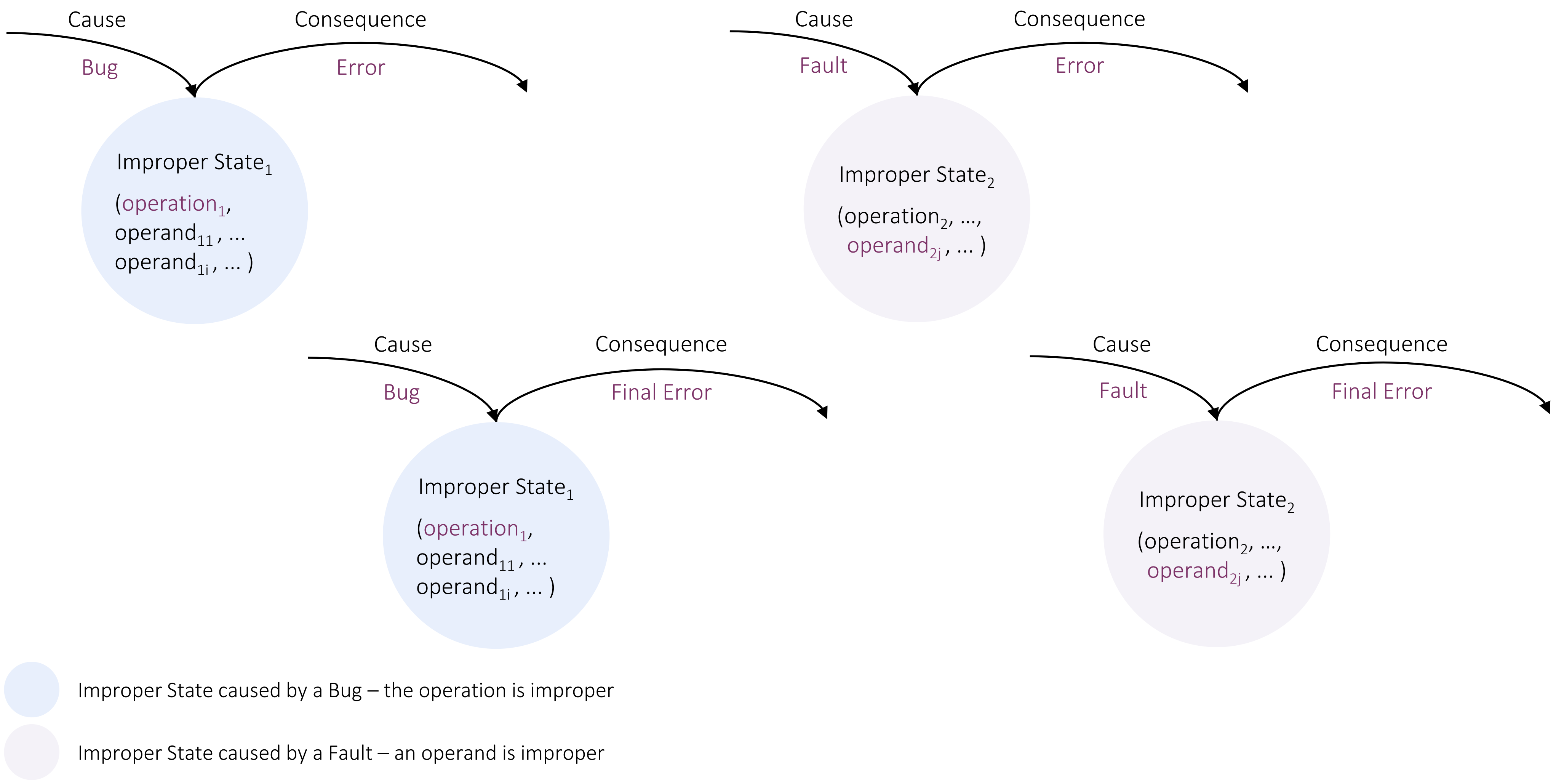
- Bugs and faults – as weakness causes
- Errors and final errors – as weakness consequences

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

BF Weakness

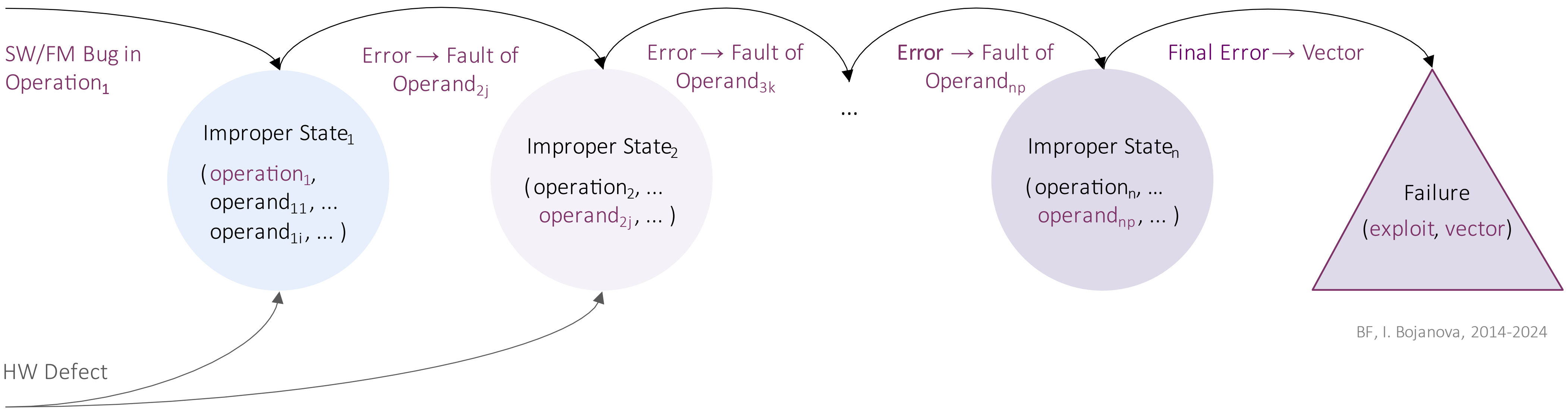

Results in Error –
Improper Operand2j
Improper State1
(operation1,
operand11 , ...
operand1i , ...)
Improper State2
(operation2, ...,
operand2j , ...)
...
Results in
Final Error
Improper Staten
(operationl, ...
operandlr , ...)
Failure
Weakness with an improper operation
Weakness with an improper operand
Weakness resulting in a final error
Failure

BF Weakness States



BF Vulnerability

BF Vulnerability



BF, I. Bojanova, 2014-2024

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

↷ Chaining

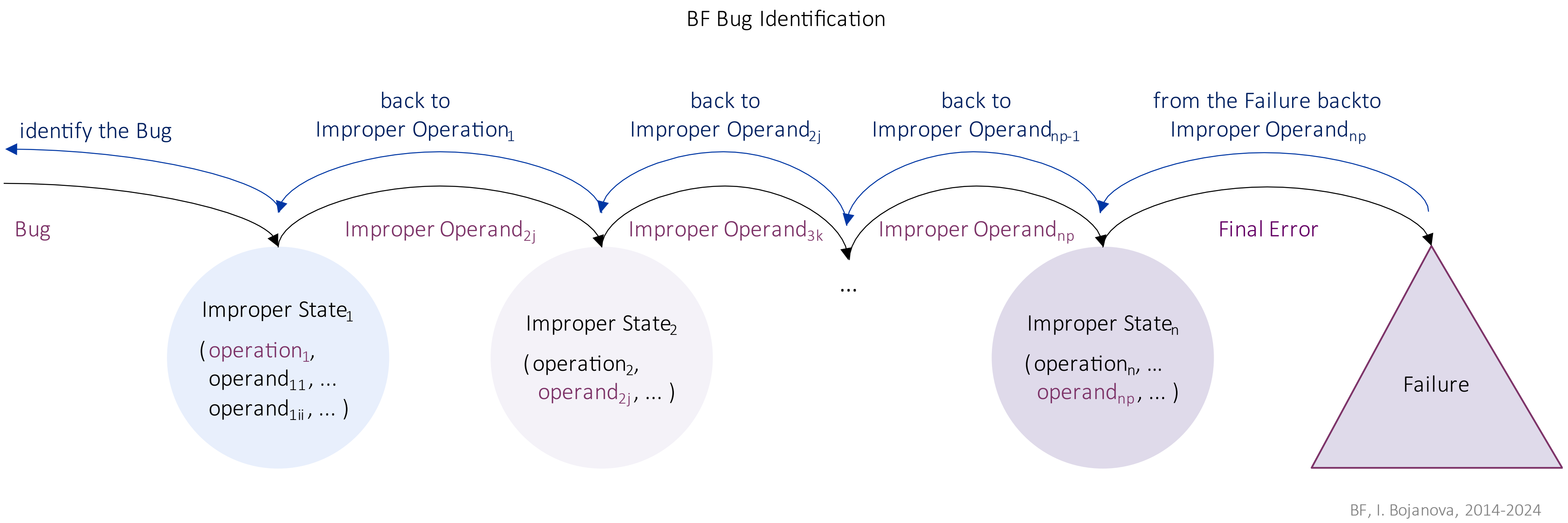
Initial State – caused by a Bug

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



Improper State: an *(operation, operand1, ..., operandn)* 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

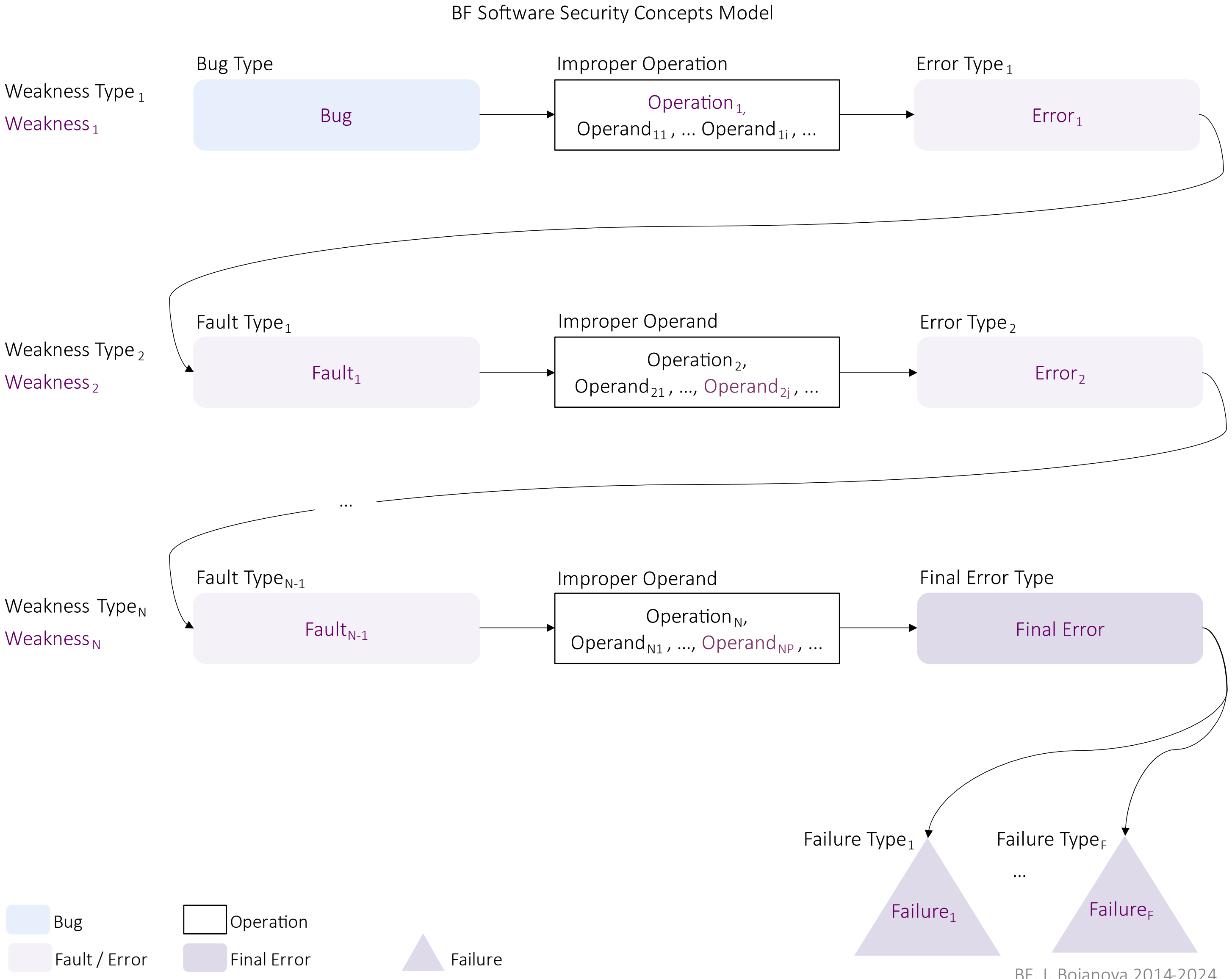
Bug/Fault – relates to Execution Phase:

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

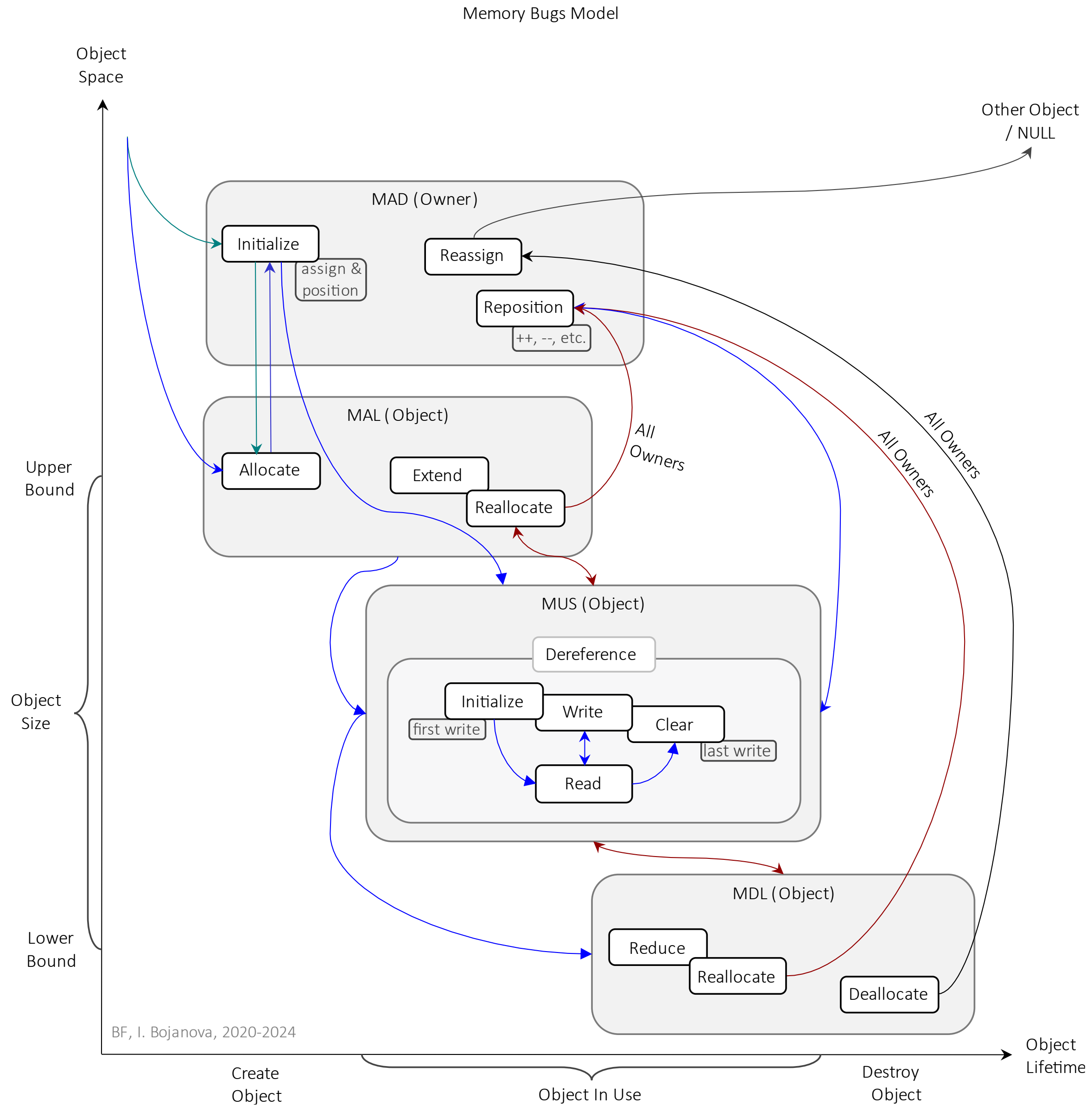


$Operation_1$ has a Bug and results in Error 1, which becomes $Fault_1$ for $Operation_2$, leading to $Error_2$.

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

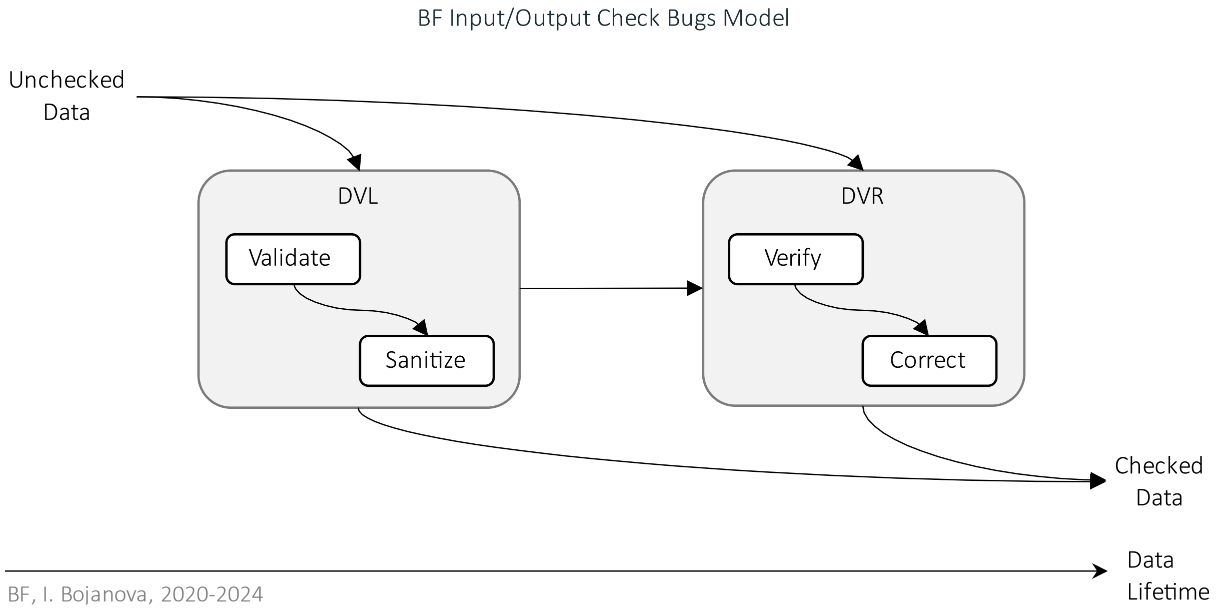
BF Bugs Models

BF Memory (_MEM) Bugs Models



- Identify Secure Code Principles:
  - Memory Safety

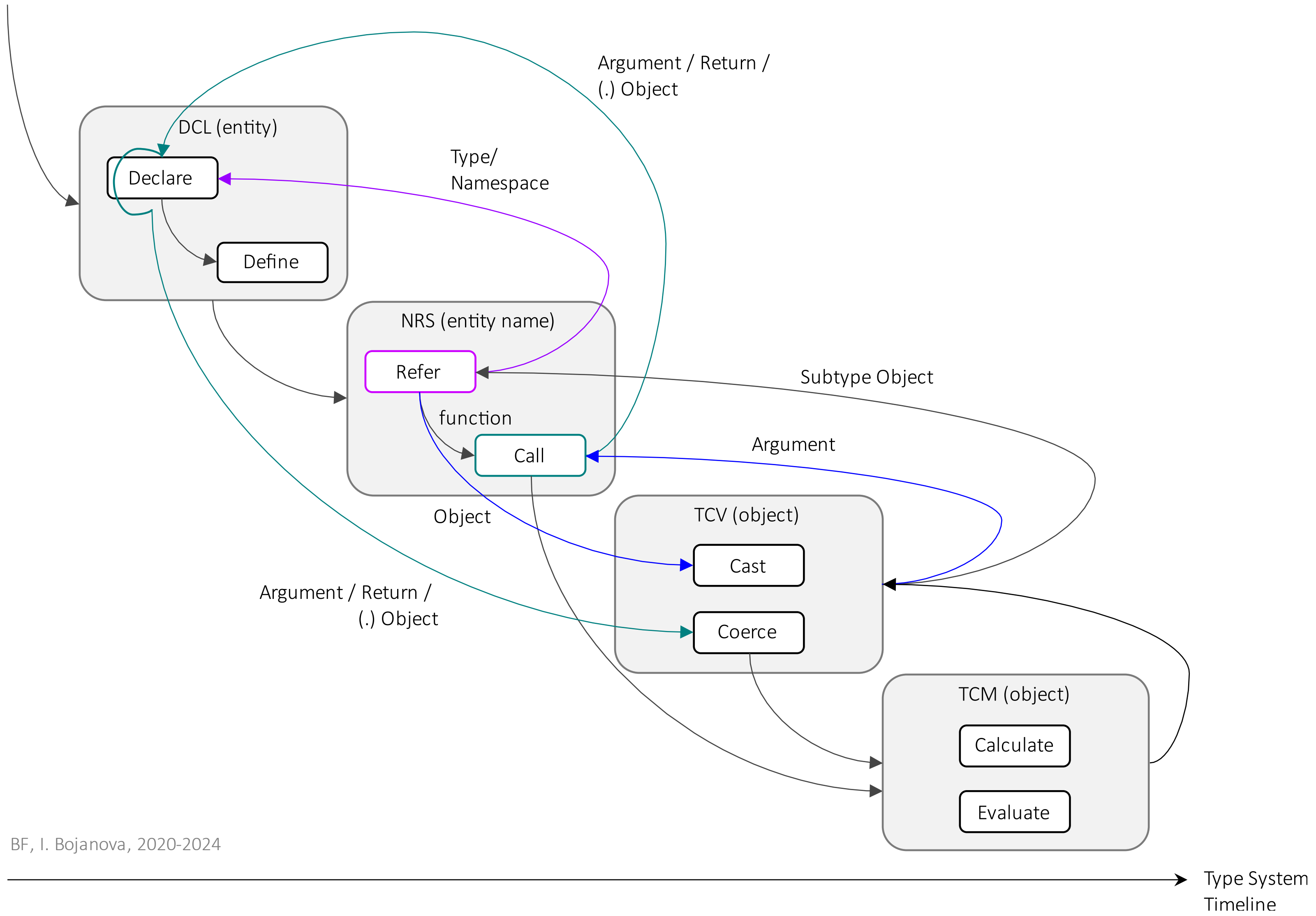
BF Input/Output Model (_INP) Bugs Model



- Identify Secure Code Principles:
  - Input/Output Safety

BF Data Type (_DAT) Bugs Model

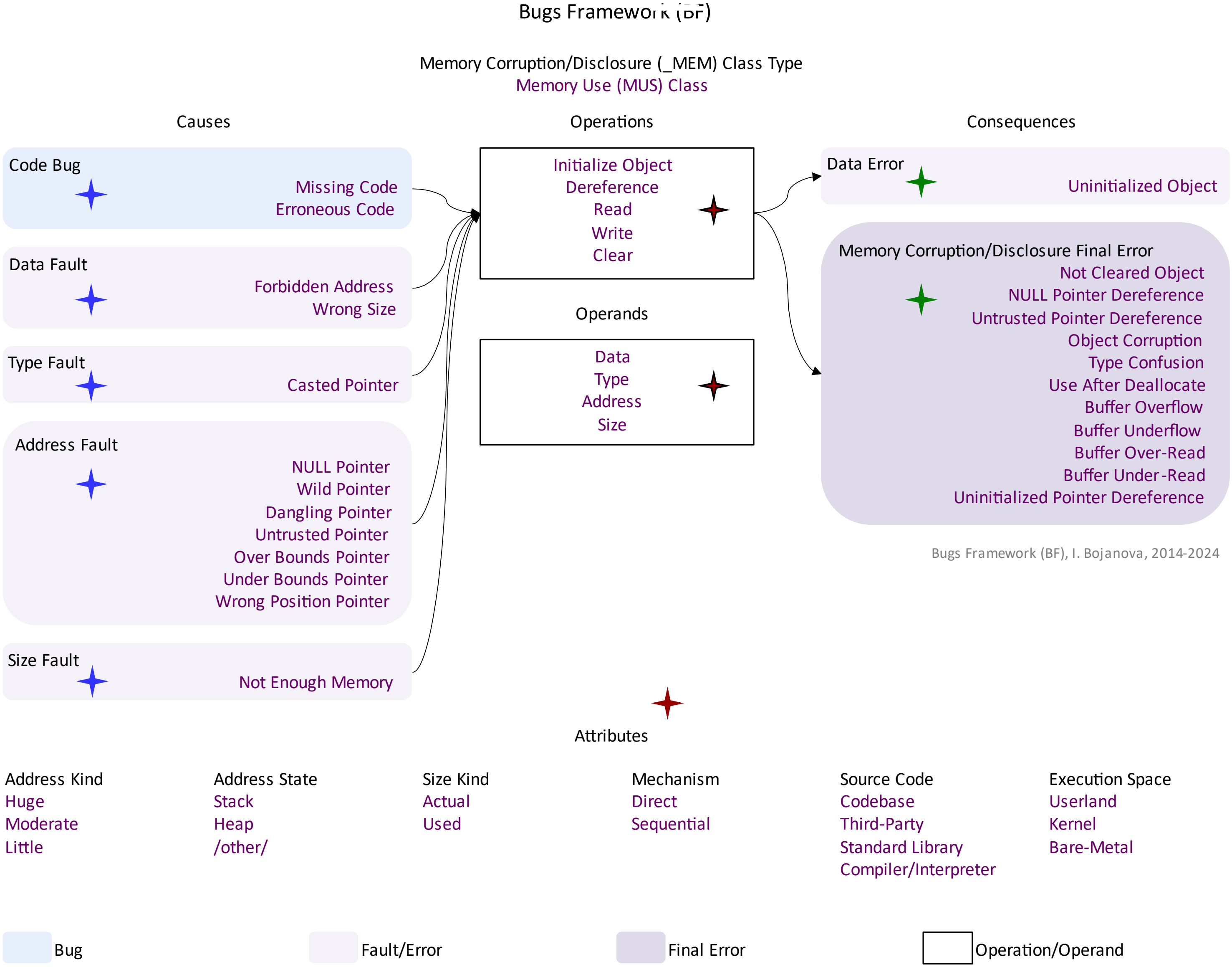
BF Data Type Bugs M odel



- Identify Secure Code Principles:
  - Data Type Safety

BF Weakness Taxonomies

BF Memory Use (MUS) Class – Example from MEM Class Type



BF Memory Use (MUS) Class

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

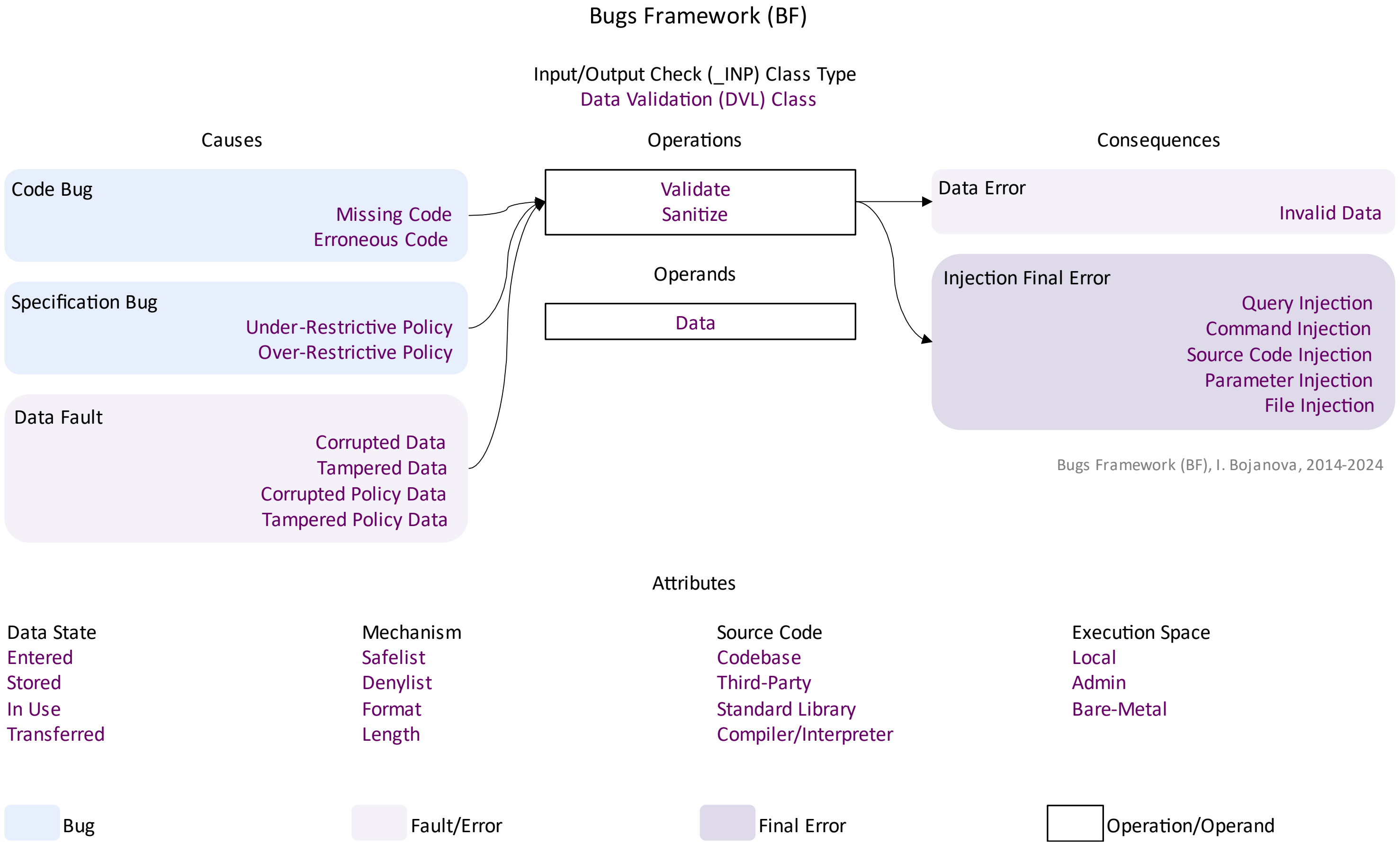
(Bug, Operation_1, Error_1) ← lookup_weakness_triple()

(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

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



BF Data Validation (DVL) Class

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

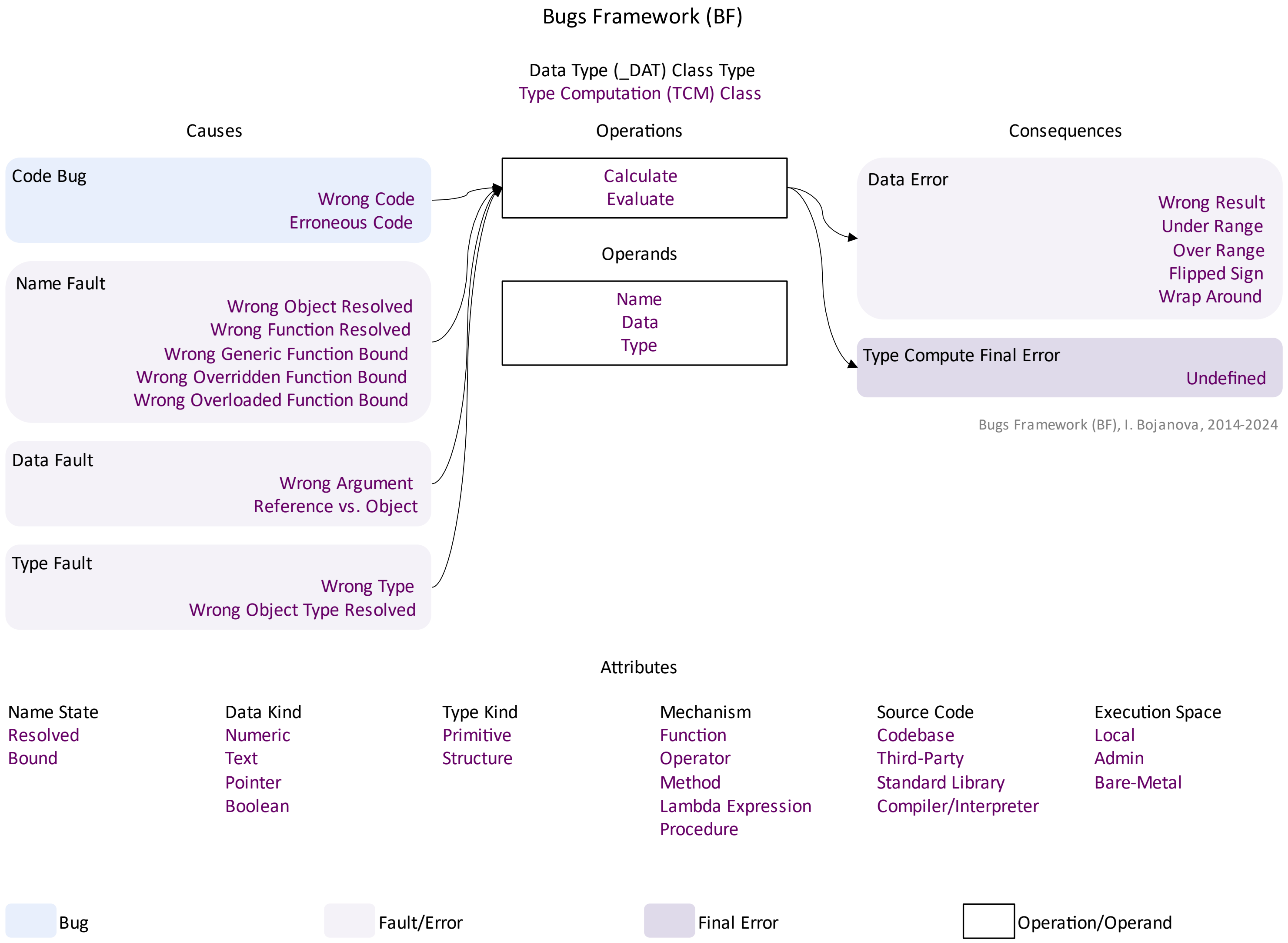
(Bug, Operation_1, Error_1) ← lookup_weakness_triple()

(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

BF Type Computation (TCM) Class – Example from _DAT Class Type



Type Computation (TCM) Class

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

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

NIST

BF.xml*

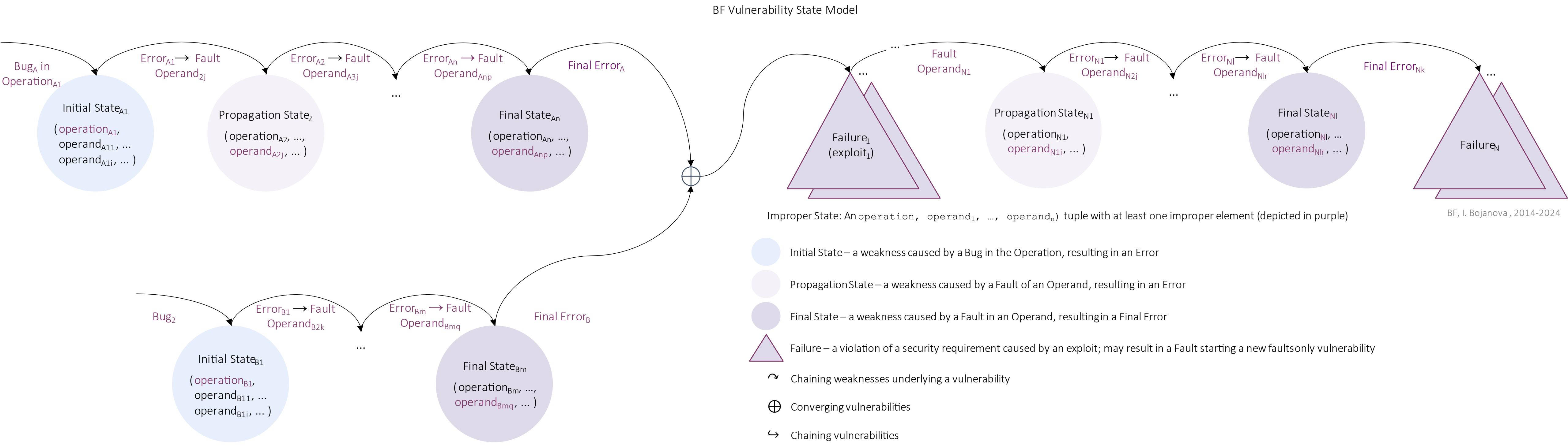
```xml
<!--@author Irena Bojanova(ivb)-->
<!--@date - 2/9/2022-->
<BF Name="Bugs Framework">
    <Cluster Name="_INP" Type="Weakness">...</Cluster>
    <Cluster Name="_DAT" Type="Weakness">
        <Class Name="DCL" Title="Declaration Bugs">
            <Operations>
                <Operation Name="Declare"/>
                <Operation Name="Define"/>
                <AttributeType Name="Mechanism">...</AttributeType>
                <AttributeType Name="Source Code">...</AttributeType>
                <AttributeType Name="Entity">...</AttributeType>
            </Operations>
            <Operands>
                <Operand Name="Type"><!--XXX-->
                    <AttributeType Name="Type Kind">...</AttributeType>
                </Operand>
            </Operands>
            <Causes>
                <BugCauseType Name="The Bug">
                    <Cause Name="Missing Code"/>
                    <Cause Name="Wrong Code"/>
                    <Cause Name="Erroneous Code"/>
                    <Cause Name="Missing Modifier"/>
                    <Cause Name="Wrong Modifier"/>
                    <Cause Name="Anonymous Scope"/>
                    <Cause Name="Wrong Scope"/>
                </BugCauseType>
            </Causes>
            <Consequences>
                <WeaknessConsequenceType Name="Improper Type (_DAT)">
                    <Consequence Name="Wrong Type"/>
```

BF.xml*

```xml
<Definitions>
    <!-- Clusters-->
    <Definition Name="_INP" Type="Weakness">Input/Output Check Bugs
    <Definition Name="_DAT" Type="Weakness">Data Type Bugs - lead t
    <Definition Name="_MEM" Type="Weakness">Memory Bugs - lead to M
    <Definition Name="_CRY" Type="Weakness">Cryptographic Store or
    <Definition Name="_RND" Type="Weakness">Random Number Generatio
    <Definition Name="_ACC" Type="Weakness">Access Control Bugs - l

    <!-- Classes - xxx update the definitions on BF web-site-->
    <!-- _INP-->
    <Definition Name="DVL">Data are validated (syntax check) or san
    <Definition Name="DVR">Data are verified (semantics check) or c
    <!-- _DAT-->
    <Definition Name="DCL">An object, a function, a type, or a name
    <Definition Name="NRS">The name of an object, a function, or a
    <Definition Name="TCV">Data are converted or coerced into other
    <Definition Name="TCM">A numeric, pointer, or string value is c
    <!-- _MEM-->
    <Definition Name="MAD">The pointer to an object is initialized,
    <Definition Name="MAL">An object is allocated, extended, or rea
    <Definition Name="MUS">An object is initialized, read, written,
    <Definition Name="MDL">An object is deallocated, reduced, or re
    ...
    <!-- Values-->
    ...
```

BF Vulnerability Models

BF Vulnerability State Model



- 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

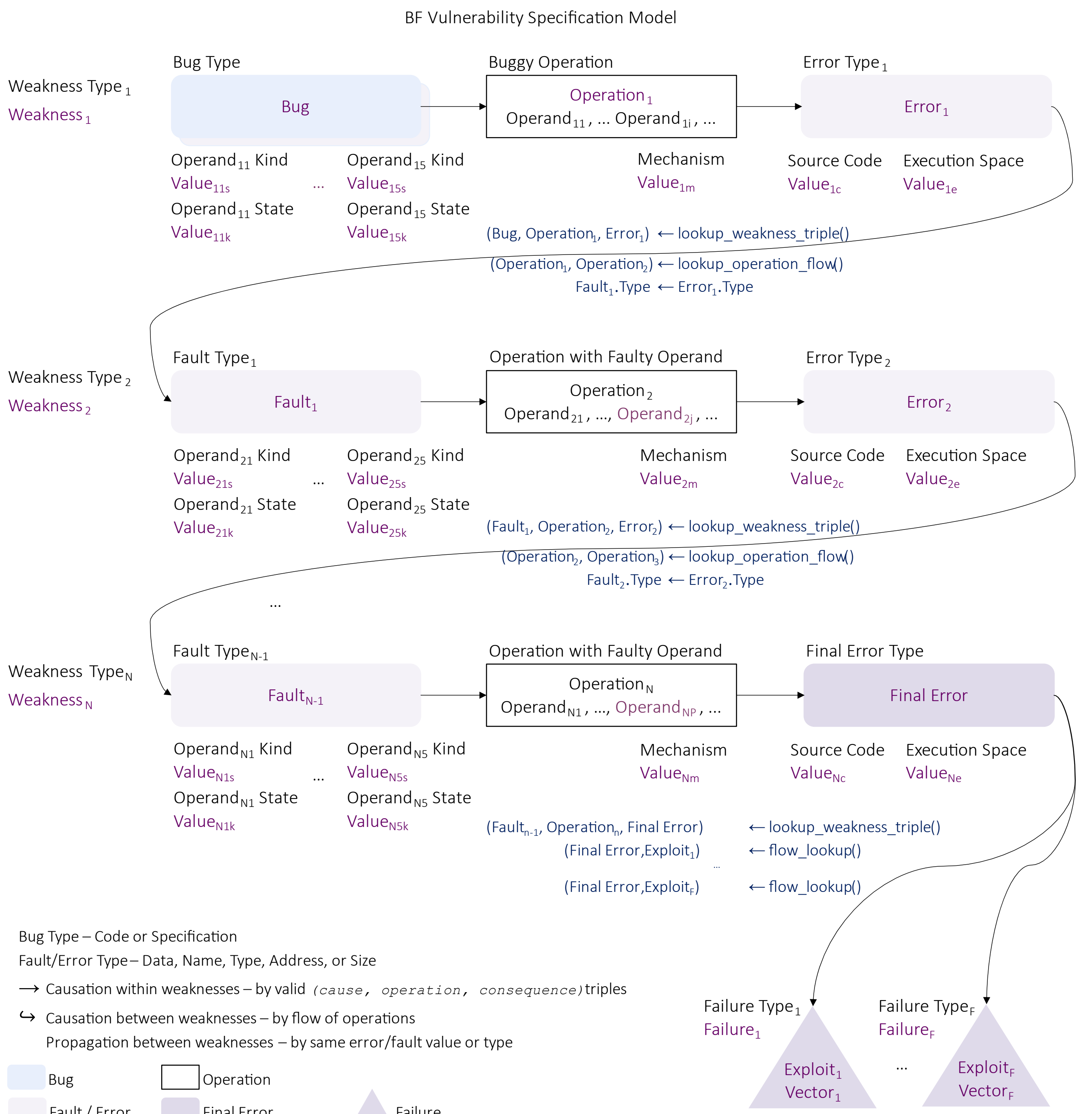
- Bug = improper operation
- Fault = improper operand

- 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 Formal Language

BF Context Free Grammar (CFG)

$$G = (V, \Sigma, R, S) \tag{1}$$

, 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)

$$\Sigma = \{\alpha \mid \alpha \in \Sigma Taxon \cup \Sigma Symbol\}$$

- 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.

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

, 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

$$\Sigma = \{\Sigma Taxon, \Sigma Symbol\} \qquad (3)$$

, where

$$\begin{aligned}\Sigma Taxon = \{&\Sigma Category, \Sigma ClassType, \Sigma Class, \Sigma BugType, \Sigma Bug,\\ &\Sigma Operation, \Sigma OperationAttributeType,\\ &\Sigma FaultType, \Sigma Fault, \Sigma OperandAttributeType, \Sigma OperandAttribute,\\ &\Sigma FinalErrorType, \Sigma FinalError\}\end{aligned}$$

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

$$ype = \{'CodeDefect', 'SpecificationDefect'\}$$

$$\begin{aligned}Bug = \{&'MissingCode', 'ErroneousCode', 'Under-RestrictivePolicy',\\ &'Over-RestrictivePolicy', 'WrongCode', 'MissingModifier',\\ &'WrongModifier', 'AnonymousScope', 'WrongScope',\\ &'MissingQualifier', 'WrongQualifier', 'MismatchedOperation', ...\}\end{aligned}$$

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

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

$$\Sigma Category = \{'Weakness', 'Failure'\}$$

$$\Sigma ClassType = \{'\_INP', '\_DAT', '\_MEM', ...\}$$

$$\Sigma Class = \{'DVL', 'DVR', 'DCL', 'NRS', 'TCV', 'TCM', 'MAD', 'MMN', 'MUS', ...\}$$

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

BF CFG Syntax

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

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

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

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

BF Syntax – LL(1) Grammar

$$S ::= Vulnerability\ Converge\_Failure \tag{6}$$

$$\begin{aligned}
Vulnerability &::= Bug_Fault\ Operation\ OperAttrs_Error_FinalError \\
Bug_Fault &::= Bug \\
&\quad |\ Fault \\
OperAttrs_Error_FinalError &::= OperationAttribute\ OperAttrs_Error_FinalError \\
&\quad |\ Error\ Fault\ OprndAttrs_Operation \\
&\quad |\ FinalError \\
OprndAttrs_Operation &::= OperandAttribute\ OprndAttrs_Operation \\
&\quad |\ Operation\ OperAttrs_Error_FinalError \\
Converge_Failure &::= \oplus\ Vulnerability\ Converge_Failure \\
&\quad |\ Vector\ Exploit\ NextVulner_Failure \\
NextVulner_Failure &::= Fault\ OprndAttrs_Operation \\
&\quad |\ Failure\ \varepsilon
\end{aligned}$$

BF Semantics – Attribute CGF

$$
\begin{aligned}
&\mathit{SyntaxRules}: \\
S &::= \mathit{Vulnerability}\ \mathit{Converge\_Failure} \\
\mathit{Vulnerability} &::= \mathit{Bug\_Fault}\ \mathit{Operation}\ \mathit{OperAttrs\_Error\_FinalError} \\
\mathit{Bug\_Fault} &::= \mathit{Bug} \\
&\quad \mid \mathit{Fault} \\
\mathit{OperAttrs\_Error\_FinalError} &::= \mathit{OperationAttribute}\ \mathit{OperAttrs\_Error\_FinalError} \\
&\quad \mid \mathit{Error}\ \mathit{Fault}_1\ \mathit{OprndAttrs\_Operation} \\
&\quad \mid \mathit{FinalError} \\
\mathit{OprndAttrs\_Operation} &::= \mathit{OperandAttribute}\ \mathit{OprndAttrs\_Operation} \\
&\quad \mid \mathit{Operation}_k\ \mathit{OperAttrs\_Error\_FinalError} \\
\mathit{Converge\_Failure} &::= \oplus\ \mathit{Vulnerability}\ \mathit{Converge\_Failure} \\
&\quad \mid \mathit{Vector}\ \mathit{Exploit}\ \mathit{NextVulner\_Failure} \\
\mathit{NextVulner\_Failure} &::= \mathit{Fault}_2\ \mathit{OprndAttrs\_Operation} \\
&\quad \mid \mathit{Failure}\ \varepsilon
\end{aligned}
\tag{7}
$$

$$
\begin{aligned}
&\mathit{SemanticRules}: \\
(\mathit{Bug},\ \mathit{Operation}_1,\ \mathit{Error}) &\leftarrow \mathit{lookup\_weakness\_triple}() \\
(\mathit{Bug},\ \mathit{Operation}_1,\ \mathit{FinalError}) &\leftarrow \mathit{lookup\_weakness\_triple}() \\
(\mathit{Fault}_1,\ \mathit{Operation}_k,\ \mathit{Error}),\ k > 1 &\leftarrow \mathit{lookup\_weakness\_triple}() \\
(\mathit{Fault}_1,\ \mathit{Operation}_k,\ \mathit{FinalError}),\ k > 1 &\leftarrow \mathit{lookup\_weakness\_triple}() \\
(\mathit{Operation}_1, \ldots \mathit{Operation}_k),\ k > 1 &\leftarrow \mathit{lookup\_operation\_flow}() \\
\mathit{Fault}_1 \leftarrow \mathit{if}\ (\mathit{Fault}_1.\mathit{ClassType} &== \mathit{Error.ClassType})\ \mathit{then}\ \mathit{Error} \\
&\mathit{Predicates}: \\
\mathit{Fault}_1.\mathit{Type} &== \mathit{Error.Type} \\
\mathit{Vector.Type} &== \mathit{FinalError.Type} \\
\mathit{Fault}_2.\mathit{Type} &== \mathit{ExploitResult.Type}
\end{aligned}
$$

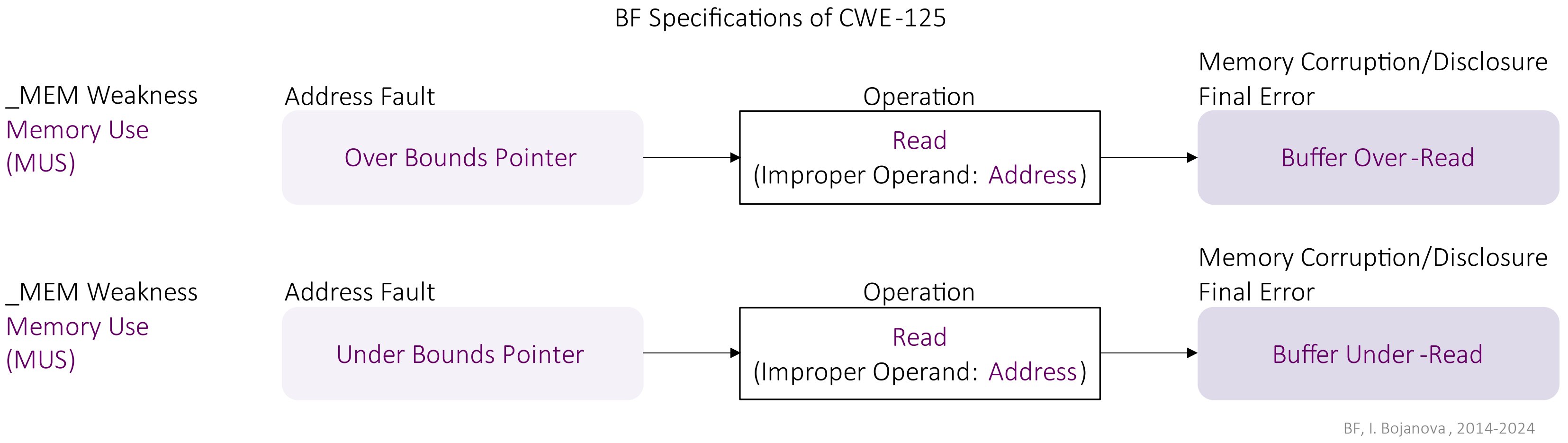
BF Specifications of CWEs

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">
```

```xml
<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>
```

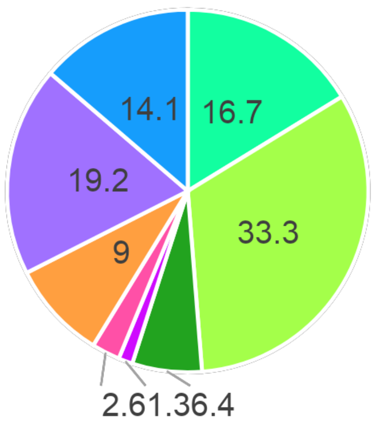
CWE-125 – Two BF Specifications

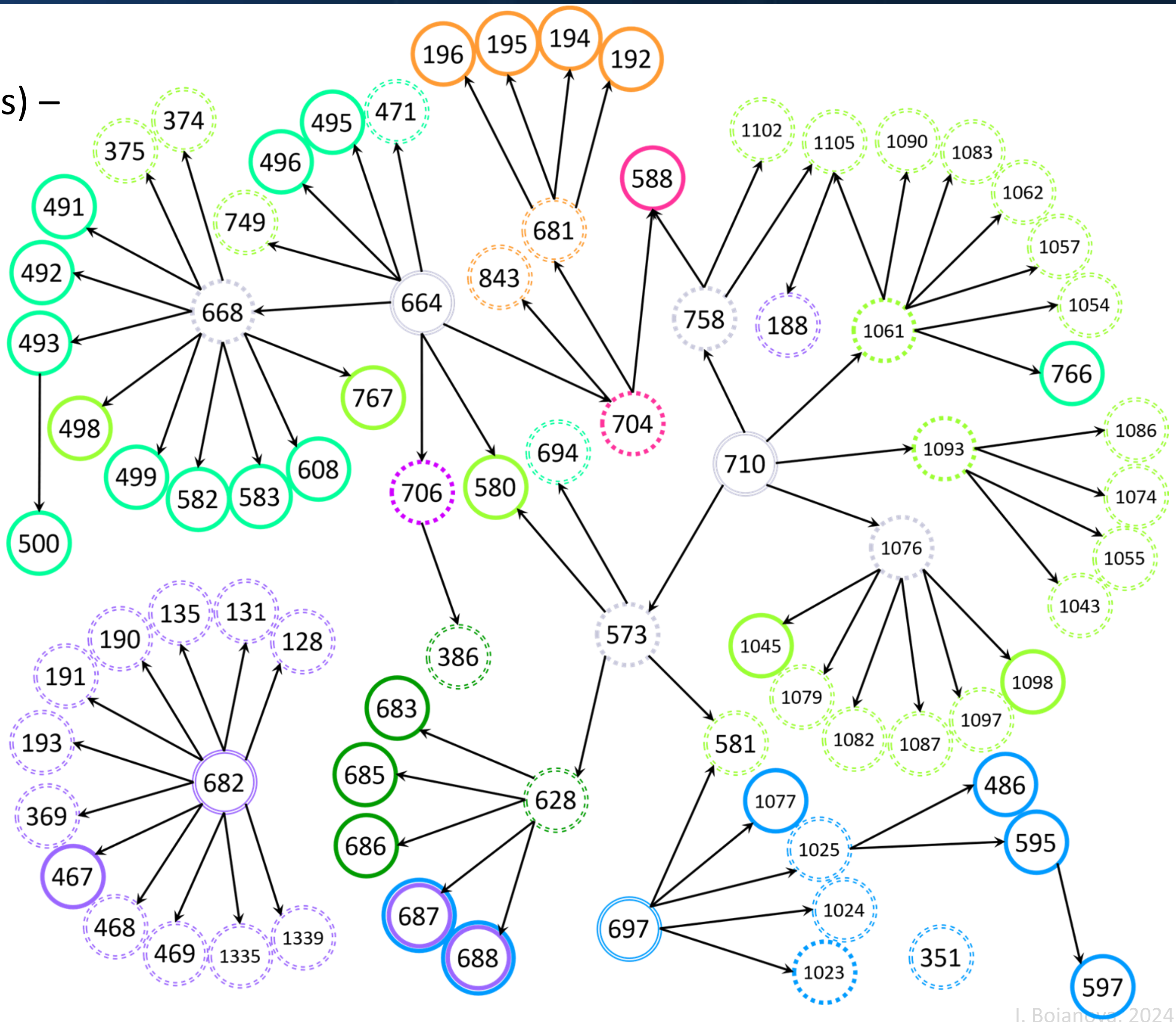


Class Type	Definition
Memory Corruption/Disclosure (_MEM)	*Bugs/weaknesses allowing a memory corruption/disclosure exploit.*
Class	Definition
Memory Use (MUS)	*An object is initialized, read, written, or cleared improperly.*
Operation	Definition
Read	*Use the value of an object's data.*
Cause	Definition
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 thelower 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





<u>CWEs by DTC, NRS, TCV, and TCM operation:</u>

DCL Declare | NRS Refer | TCV Cast | TCM Calculate

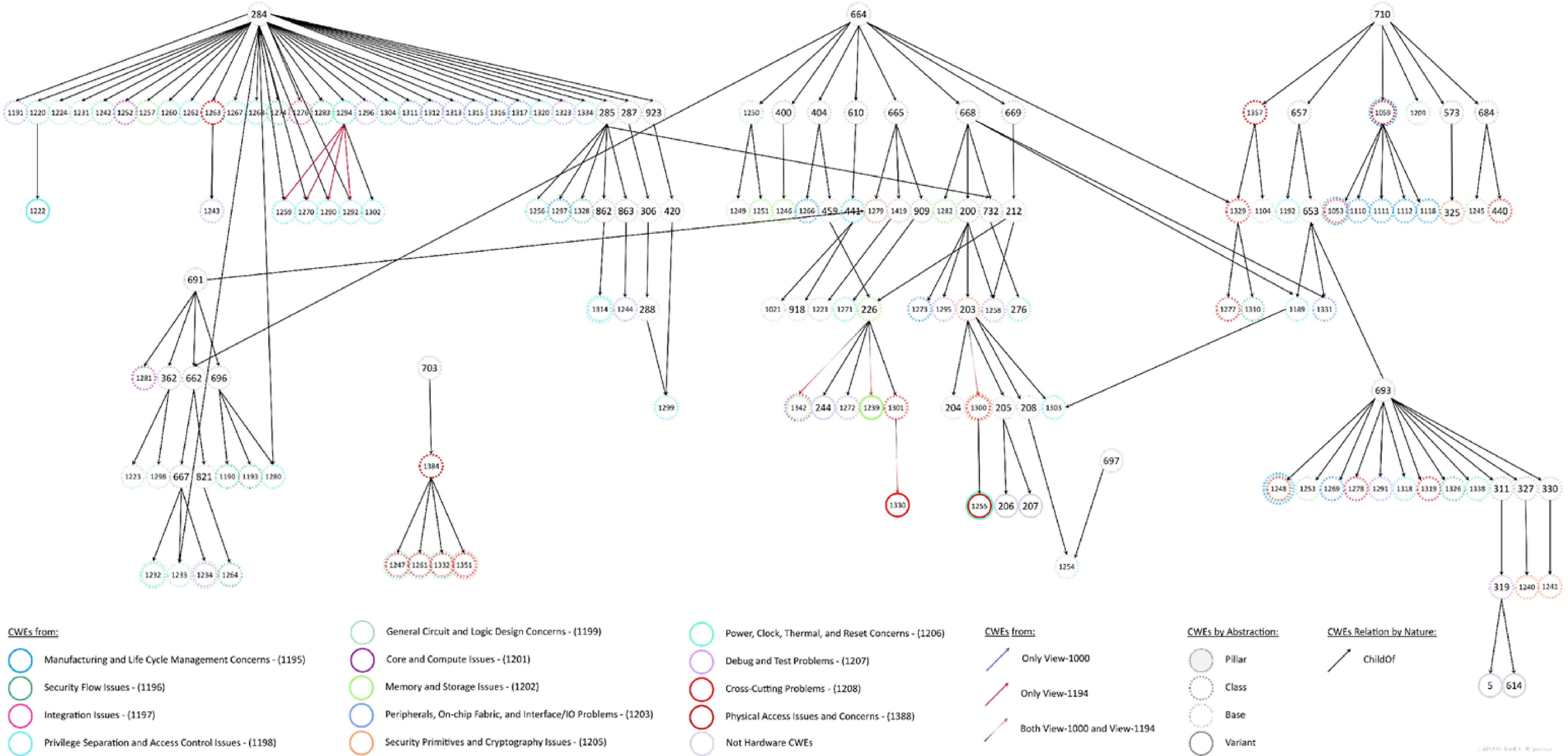
DCL Define | NRS Call | TCV Coerce | TCM Evaluate

<u>CWEs by Abstraction:</u>

Pillar | Base

Class | Variant

Analyzing HW CWEs



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

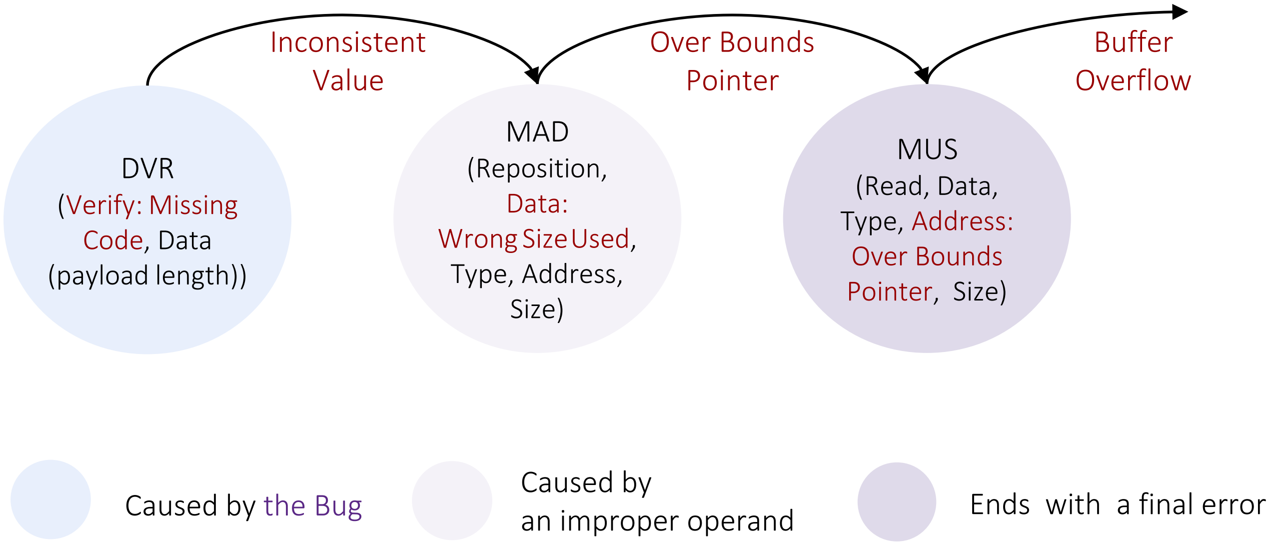
Heartbleed (CVE-2014-0160)

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.

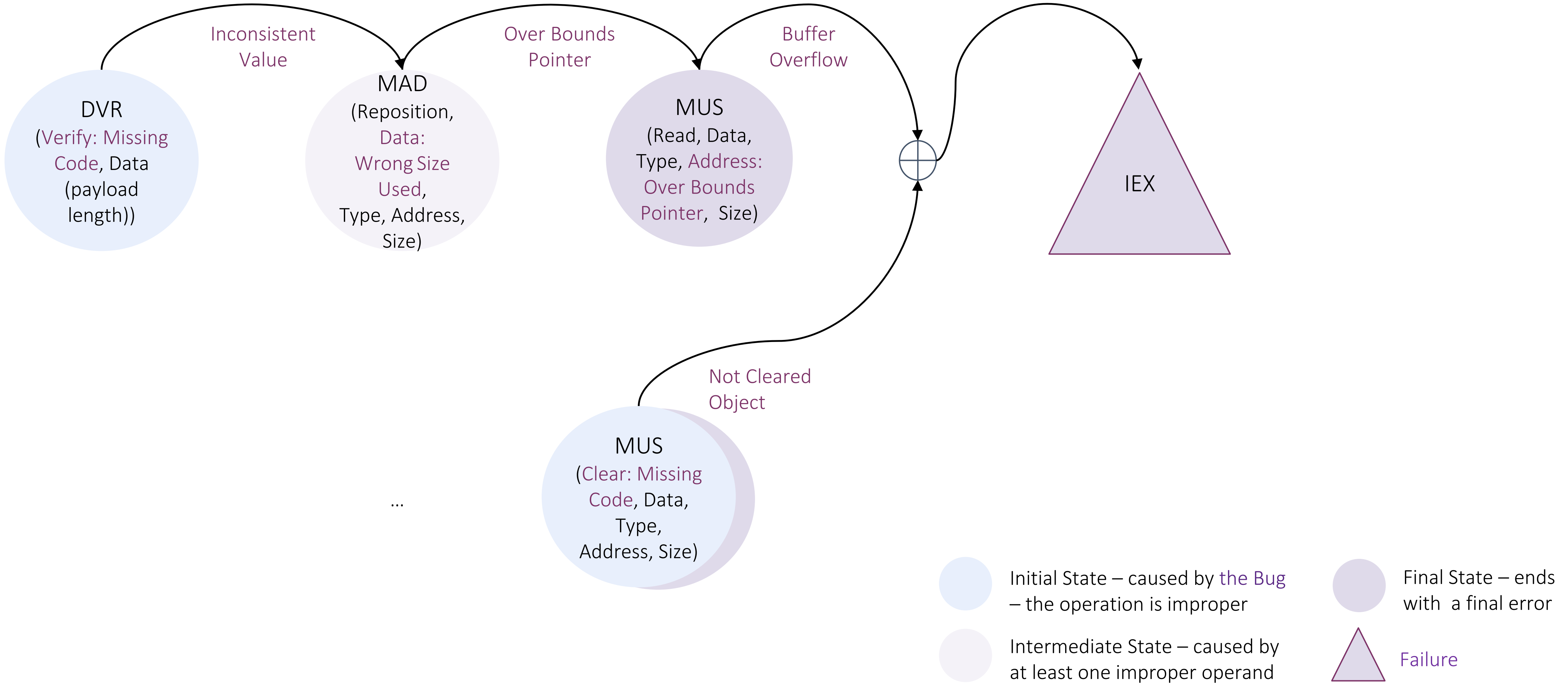
```
dtls1_process_heartbeat(SSL *s)
   {unsigned char *p = &s->s3->rrec.data[0], *pl;
   unsigned short hbtype;
   unsigned
   int payload;
   unsigned int padding = 16; /* Use minimum padding */

   /* Read type and payload length first */
   hbtype = *p++;
   n2s(p, payload);
   pl = p;
...
   if (hbtype == TLS1_HB_REQUEST)
       {
       unsigned char *buffer, *bp;
...
       /* Allocate memory for the response, size is 1 byte
       * message type, plus 2 bytes payload, plus
       * payload, plus padding
       */
       buffer = OPENSSL_malloc(1 + 2 + payload + padding);
       bp = buffer;

       /* Enter response type, length and copy payload */
       *bp++ = TLS1_HB_RESPONSE;
       s2n(payload, bp);
       memcpy(bp, pl, payload);
```

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

payload, bp, pl

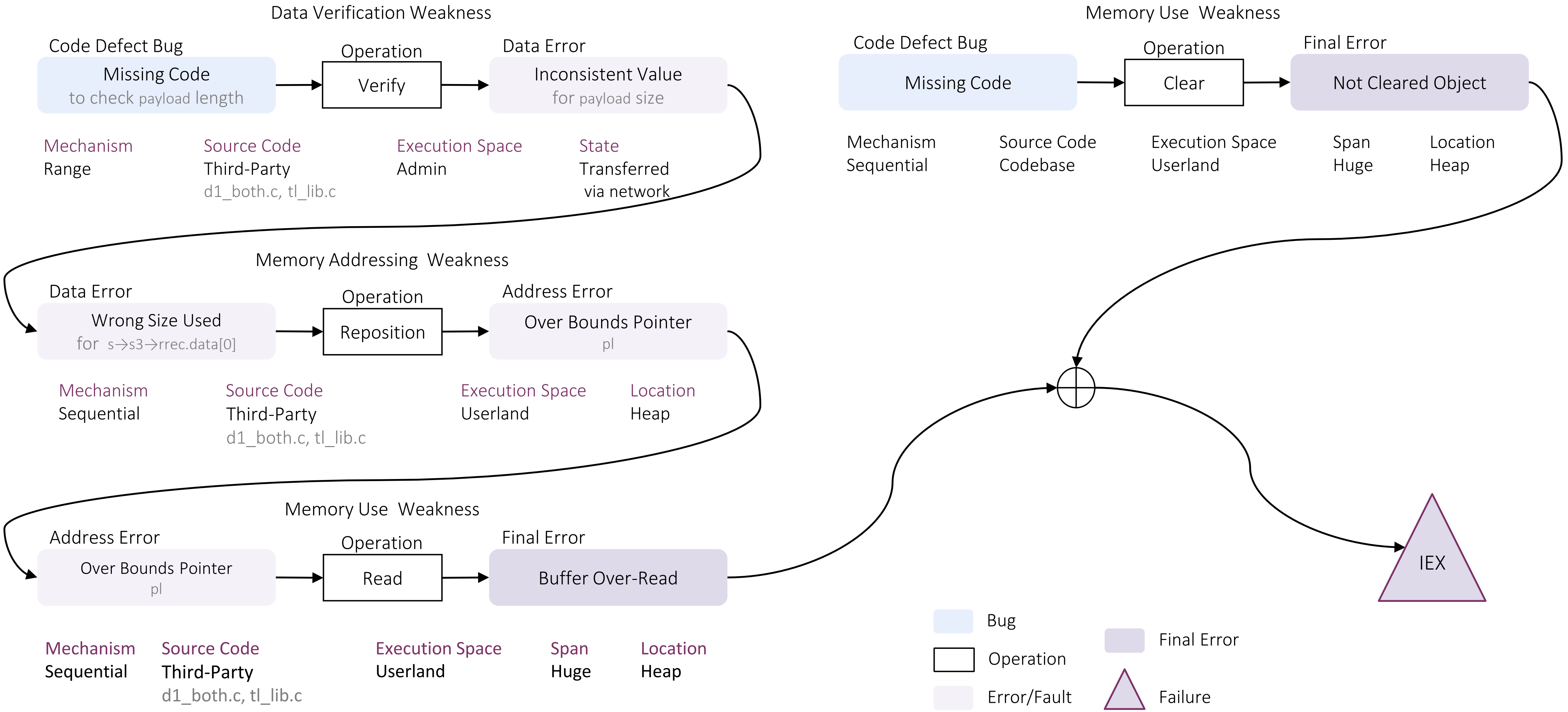


BF States of CVE-2014-0160 (Heartbleed)

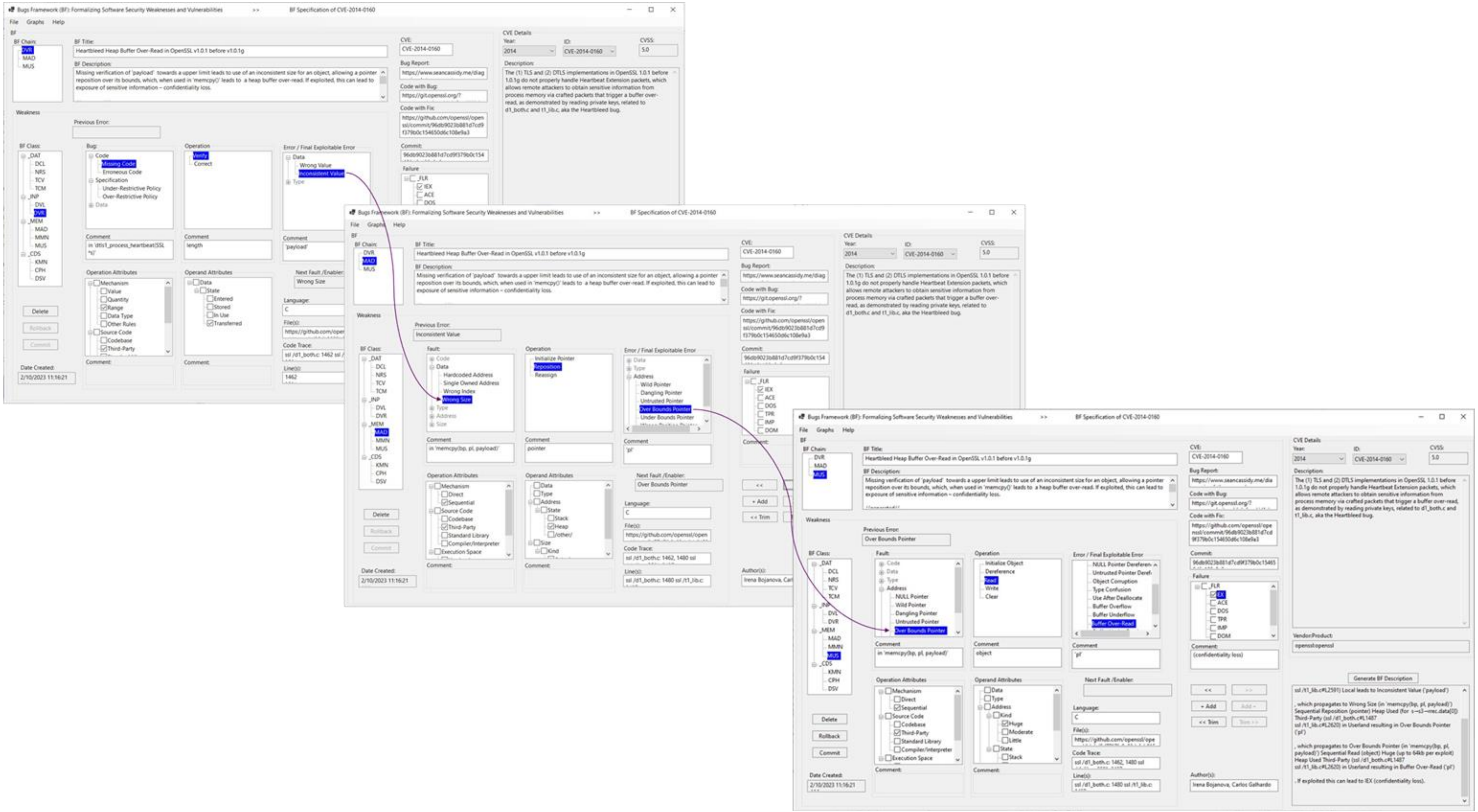


I. Bojanova, 2024

BF Specification of CVE-2014-0160 (Heartbleed)



BF Tool – BF Specification of Heartbleed



CVE-2014-0160 - Heartbleed.bfcve

CVE-2014-016...rtbleed.bfcve

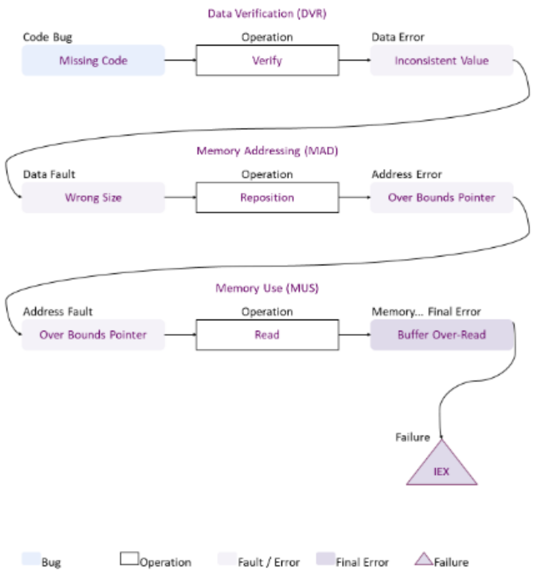

```xml
<?xml version="1.0" encoding="utf-8"?>
<CVE Name="1 CVE-2014-0160">
    <BugWeakness Type="_INP" Class="DVR">
        <Cause Type="The Bug">Missing Code</Cause>
        <Operation>Verify</Operation>
        <Consequence Comment="for payload size" Type="Improper Data">Inconsistent Value</Consequence>
        <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>
```

BFCVE Dataset

BF Welcome! Models Formalism Tools APIs Approach Background Publications Contact Search BF.. Ctrl + /

> TAXONOMY
BF CVE
Overview
CVE-2004-1287
CVE-2006-2362
CVE-2007-1320
CVE-2007-6429
CVE-2008-4539
CVE-2013-4930
CVE-2013-4934
CVE-2014-0160
CVE-2015-0235
CVE-2015-5221
CVE-2017-17833
CVE-2018-14557
CVE-2019-14814
CVE-2021-21834
CVE-2022-34835
CVE-2023-1283
CVE-2023-2356
CVE-2023-2564
CVE-2023-3765
...

BF Specification of CVE-2014-0160 Heartbleed Heap Buffer Over-Read in OpenSSL v1.0.1 before v1.0.1g



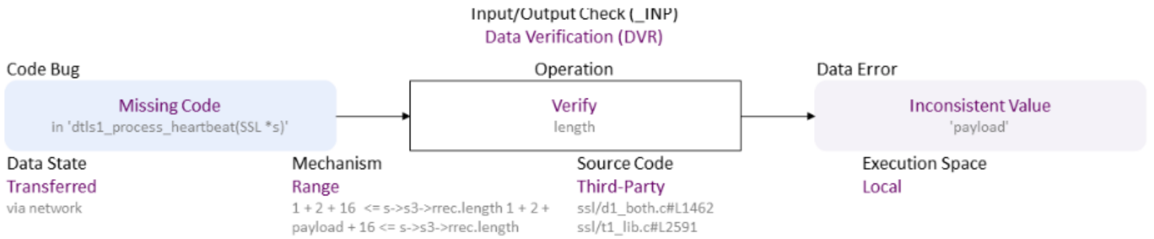
Missing verification of 'payload' towards a upper limit leads to use of an inconsistent size for an object, allowing a pointer reposition over its bounds, which, when used in 'memcpy()' leads to a heap buffer over-read. If exploited, this can lead to exposure of sensitive information – confidentiality loss.

//generated// Missing Code (in 'dtls1_process_heartbeat(SSL *s)') to Range Verify length (1 + 2 + 16 <= s->s3->rrec.length 1 + 2 + payload + 16 <= s->s3->rrec.length) Transferred (via network) in Third-Party (ssl /d1_both.c#L1462 ssl /t1_lib.c#L2591) Local leads to Inconsistent Value ('payload')

, which propagates to Wrong Size (in 'memcpy(bp, pl, payload)') Sequential Reposition (pointer) Heap Used (for s→s3→rrec.data[0]) Third-Party (ssl /d1_both.c#L1487 ssl /t1_lib.c#L2620) in Userland resulting in Over Bounds Pointer ('pl')

, which propagates to Over Bounds Pointer (in 'memcpy(bp, pl, payload)') Sequential Read (object) Huge (up to 64kb per exploit) Heap Used Third-Party (ssl /d1_both.c#L1487 ssl /t1_lib.c#L2620) in Userland resulting in Buffer Over-Read ('pl')

. If exploited this can lead to IEX (confidentiality loss).



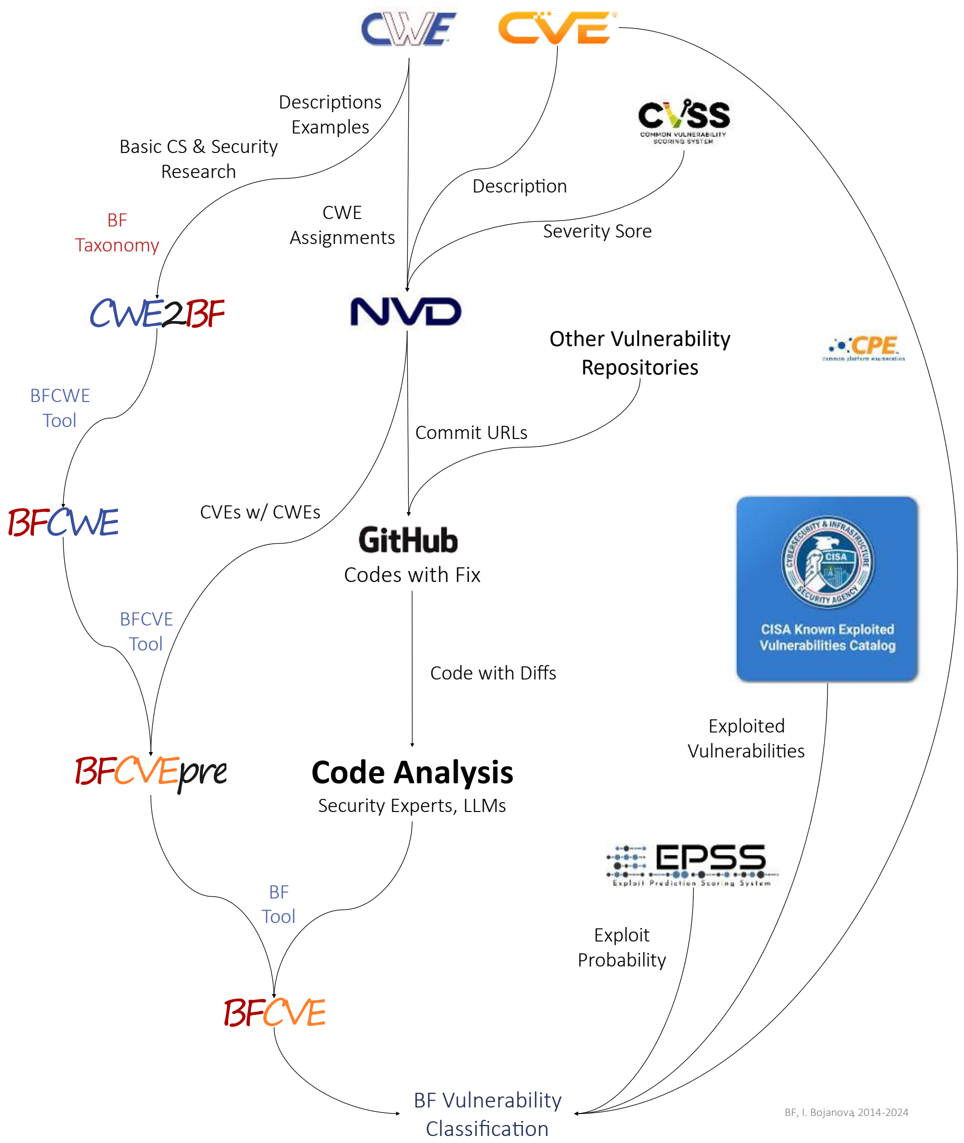
ssl/d1_both.c#L1462
ssl/t1_lib.c#L2591

Show/Hide Definitions | vendor:product: openssl:openssl | Bug Report | Code with Bug | Code with Fix | NVD Entry

Class	Definition
DVR	Data Verification (DVR) class – Data are verified (semantics check) or corrected (assign, remove) improperly.
...	

BF Vulnerability Classification

BF Vulnerability Classification Model




```sql
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
```

BF Data in NVD

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/

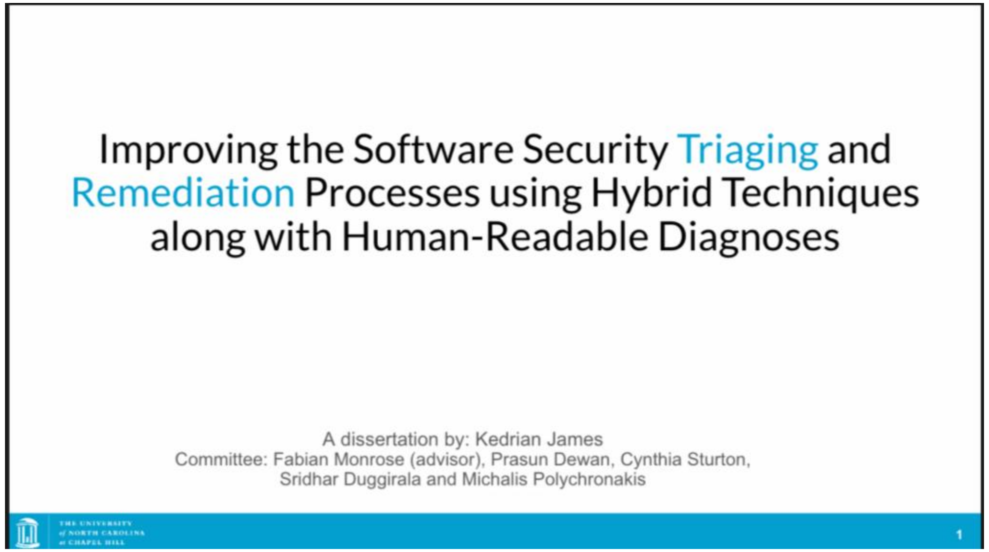
BF in Security Research

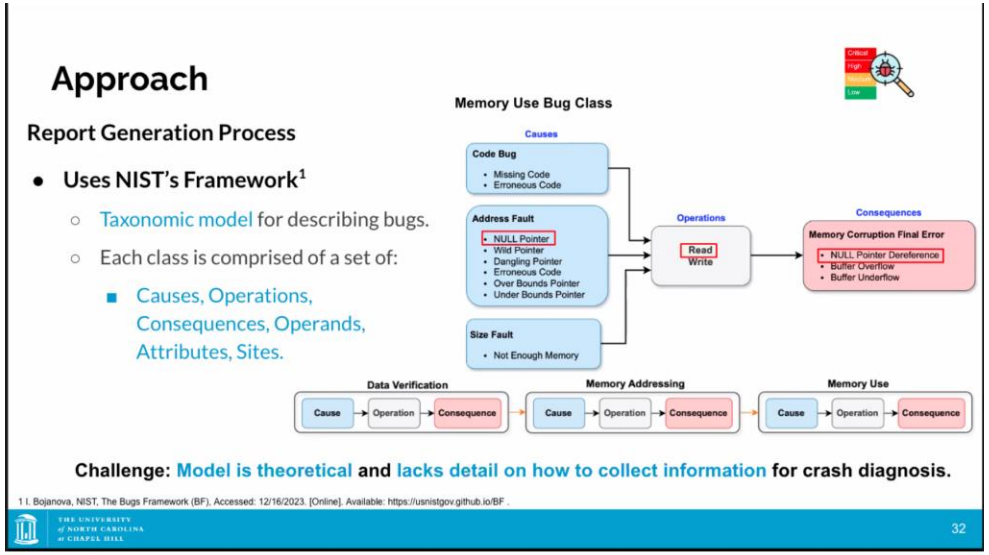
Machine readable formats of:

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

✓ Projects related to:

  - Vulnerability specification generation
  - Bug detection
  - Vulnerability analysis and remediation
  - Security failures and risks

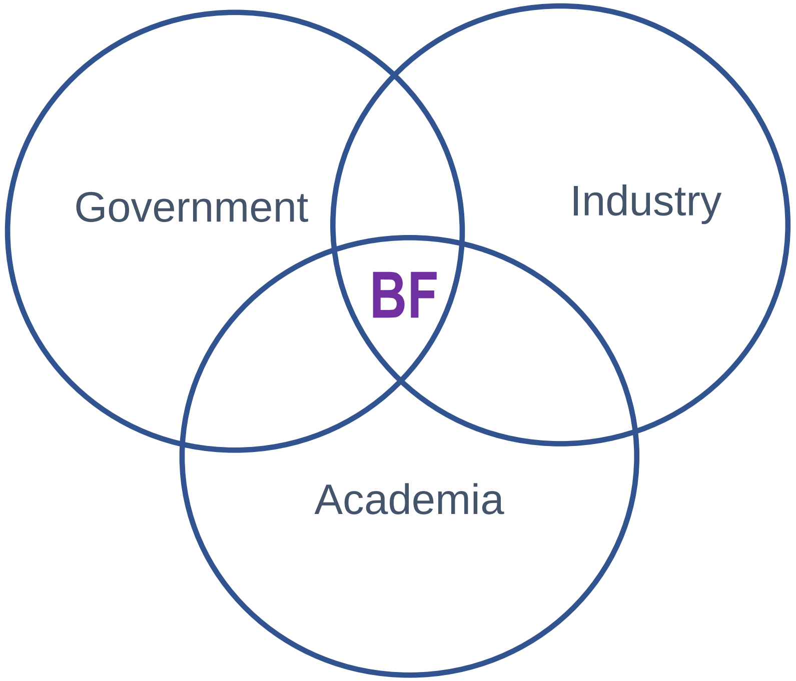




BF – Potential Impact

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/