Bugs Framework (BF)

November 8, 2023

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

Irena Bojanova

https://samate.nist.gov/BF

Agenda

- Introduction

Security Concepts

“BadAlloc” Pattern

- BF

Bugs Models

Weakness Taxonomy

Vulnerability Model

Formal Language

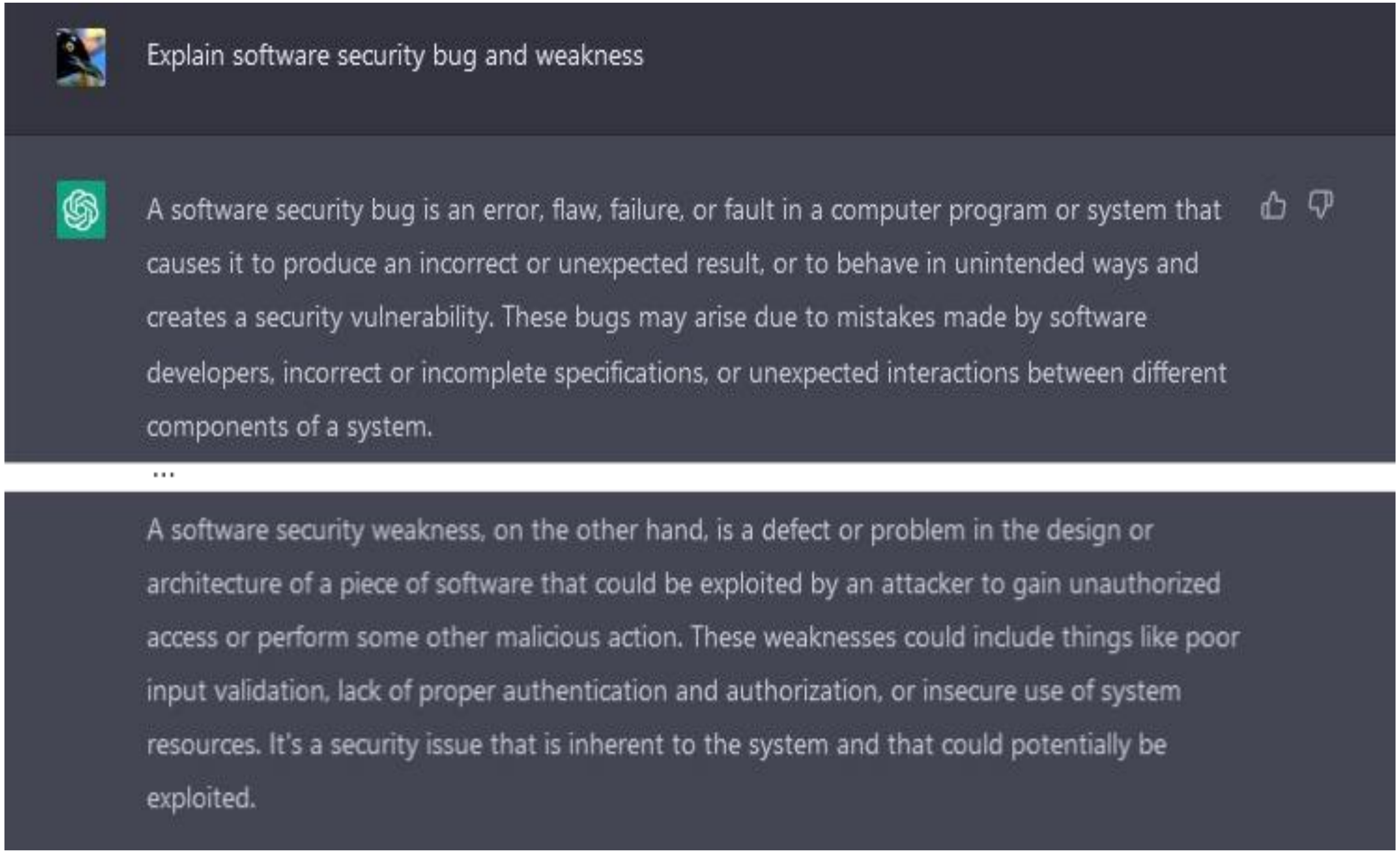
Generation Tools

- BF Hands On:
  - BF Specifications of Vulnerabilities
  - NLP, ML, AI Applications
- Potential Impacts

Introduction

Software Security Vulnerabilities

- Lead to failures – used to attack:
  - Cyberspace
  - Critical infrastructure
- Loosely used Notions:
  - Bug
  - Fault
  - Error
  - Weakness
  - Vulnerability
  - Failure
- AI chatbots generate conflating explanations



Explain software security bug and weakness

A software security bug is an error, flaw, failure, or fault in a computer program or system that causes it to produce an incorrect or unexpected result, or to behave in unintended ways and creates a security vulnerability. These bugs may arise due to mistakes made by software developers, incorrect or incomplete specifications, or unexpected interactions between different components of a system.

...

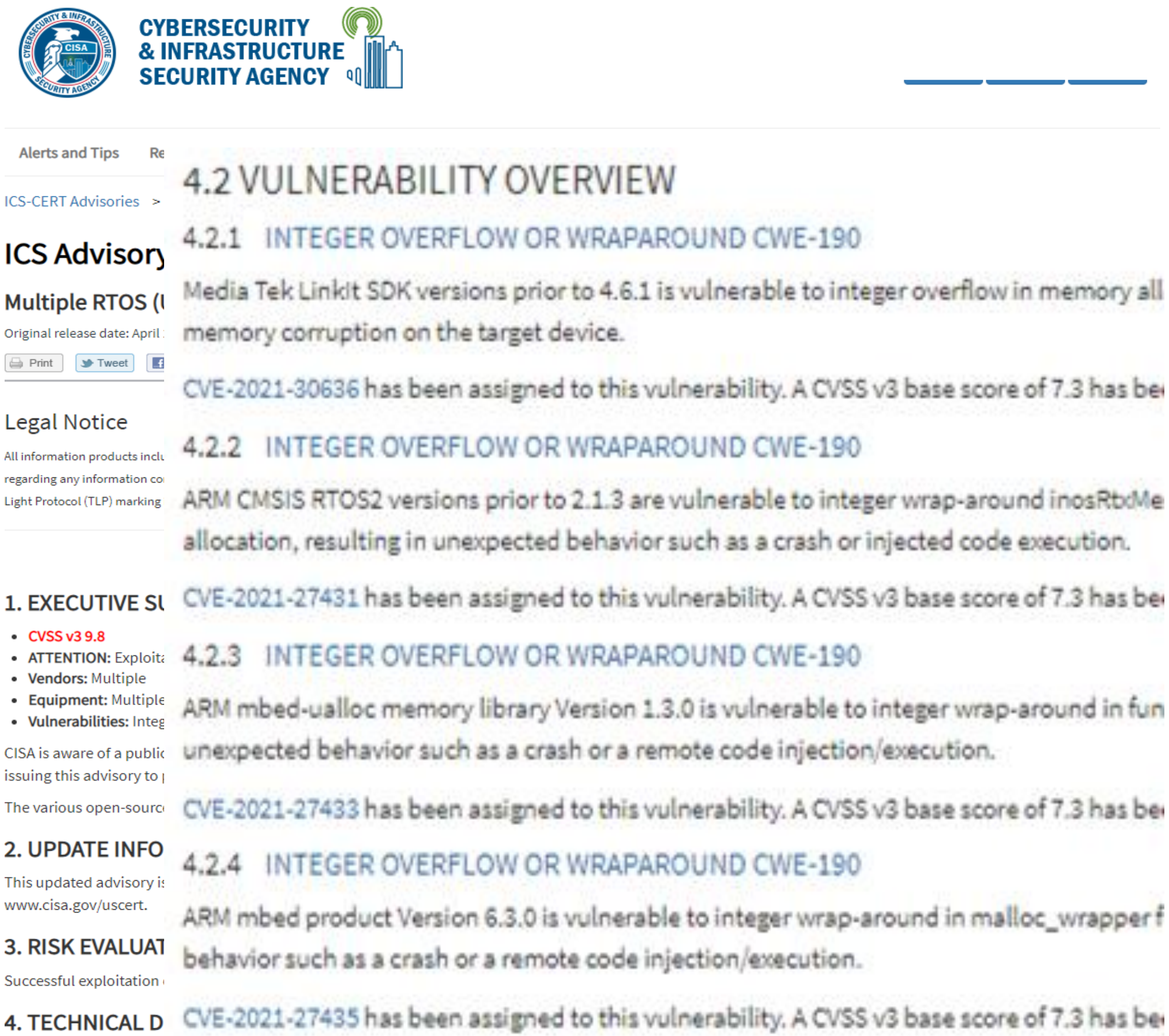
A software security weakness, on the other hand, is a defect or problem in the design or architecture of a piece of software that could be exploited by an attacker to gain unauthorized access or perform some other malicious action. These weaknesses could include things like poor input validation, lack of proper authentication and authorization, or insecure use of system resources. It's a security issue that is inherent to the system and that could potentially be exploited.

Definitions

- Software Security Vulnerability
  - A chain of weaknesses linked by causality
  - Starts with a bug
  - Ends with a final error, which if exploited leads to a security failure
- Software Security Weakness
  - A `(bug, operation, error)` or `(fault, operation, error)` triple.
  - An instance of a weakness type that relates to a distinct phase of software execution, the operations specific for that phase and the operands required as input to those operations.
- Software Security Bug
  - A code or specification defect (operation defect)
- Software Fault
  - A name, data, type, address, or size error (operand error)
- Software Error
  - The result from an operation with a bug or a faulty operand
  - Becomes a next fault or is a final error
- Software Final Error
  - An exploitable or undefined system behavior
  - Leads to a security failure
- Security Failure
  - A violation of a system security requirement

Bugs Framework (BF)
https://samate.nist.gov/BF

"BadAlloc" Pattern – 25 CVEs



CYBERSECURITY & INFRASTRUCTURE SECURITY AGENCY

Alerts and Tips Re

ICS-CERT Advisories >

ICS Advisory

Multiple RTOS (

Original release date: April

Print Tweet

Legal Notice

All information products inclu
regarding any information co
Light Protocol (TLP) marking

1. EXECUTIVE SU

- CVSS v3 9.8
- **ATTENTION:** Exploita
- **Vendors:** Multiple
- **Equipment:** Multiple
- **Vulnerabilities:** Integ

CISA is aware of a public
issuing this advisory to p

The various open-sourc

2. UPDATE INFO

This updated advisory is
www.cisa.gov/uscert.

3. RISK EVALUAT

Successful exploitation

4. TECHNICAL D

4.2 VULNERABILITY OVERVIEW

4.2.1 INTEGER OVERFLOW OR WRAPAROUND CWE-190

Media Tek LinkIt SDK versions prior to 4.6.1 is vulnerable to integer overflow in memory all
memory corruption on the target device.

CVE-2021-30636 has been assigned to this vulnerability. A CVSS v3 base score of 7.3 has be

4.2.2 INTEGER OVERFLOW OR WRAPAROUND CWE-190

ARM CMSIS RTOS2 versions prior to 2.1.3 are vulnerable to integer wrap-around inosRtxMe
allocation, resulting in unexpected behavior such as a crash or injected code execution.

CVE-2021-27431 has been assigned to this vulnerability. A CVSS v3 base score of 7.3 has be

4.2.3 INTEGER OVERFLOW OR WRAPAROUND CWE-190

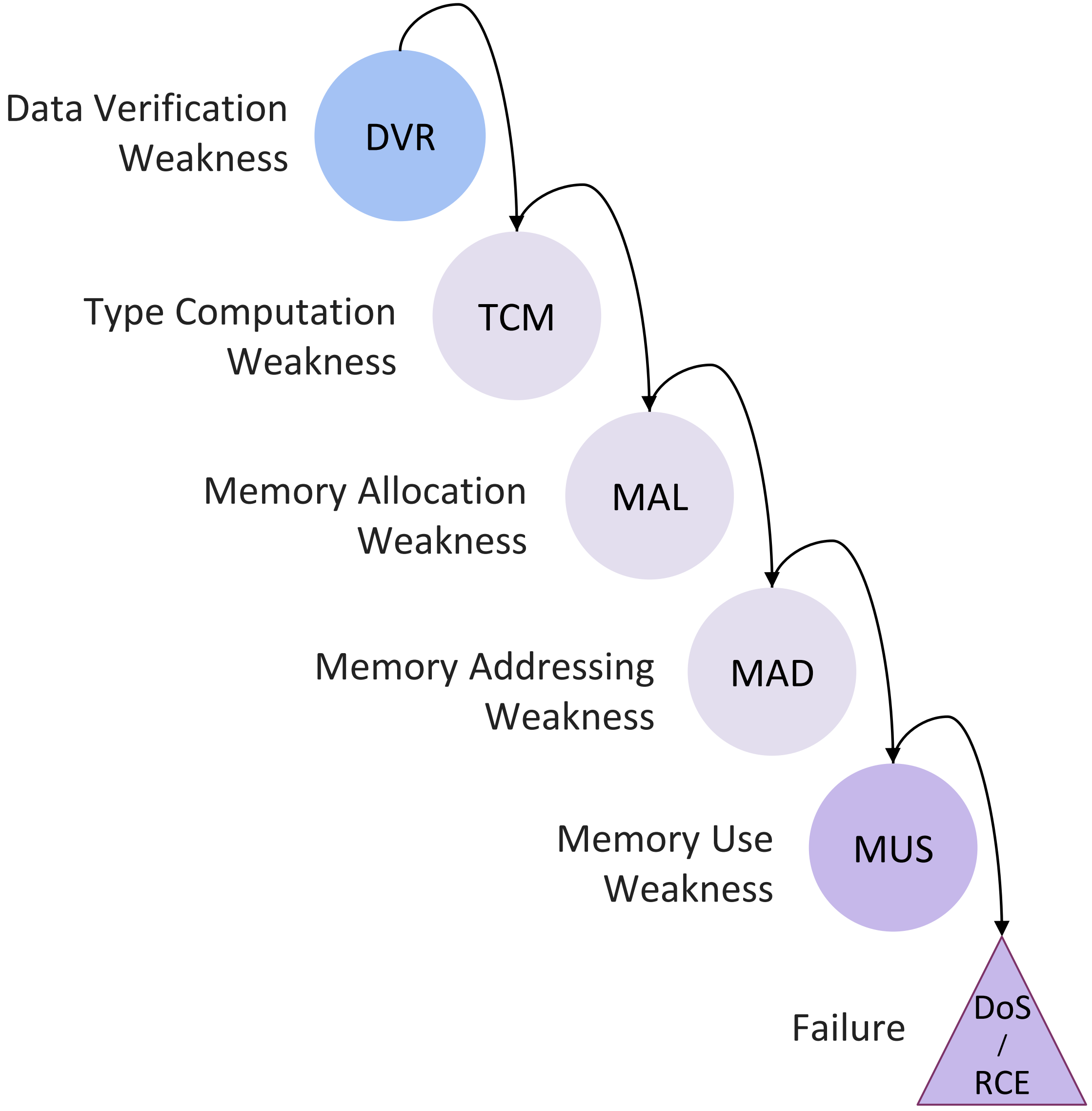
ARM mbed-ualloc memory library Version 1.3.0 is vulnerable to integer wrap-around in fun
unexpected behavior such as a crash or a remote code injection/execution.

CVE-2021-27433 has been assigned to this vulnerability. A CVSS v3 base score of 7.3 has be

4.2.4 INTEGER OVERFLOW OR WRAPAROUND CWE-190

ARM mbed product Version 6.3.0 is vulnerable to integer wrap-around in malloc_wrapper f
behavior such as a crash or a remote code injection/execution.

CVE-2021-27435 has been assigned to this vulnerability. A CVSS v3 base score of 7.3 has be



Data Verification Weakness — DVR

Type Computation Weakness — TCM

Memory Allocation Weakness — MAL

Memory Addressing Weakness — MAD

Memory Use Weakness — MUS

Failure — DoS / RCE

BF Defined

BF Definition

BF is a classification system of software security bugs, faults, and weaknesses
that allows unambiguous formal descriptions of
software security vulnerabilities that exploit them.

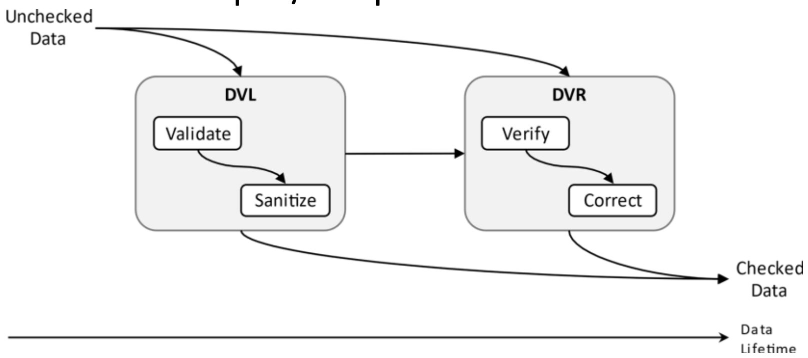
BF comprises:

- Bugs models
- A weakness taxonomy
- A vulnerability model
- An LL(1) formal language
- A relational database
- Generation tools.

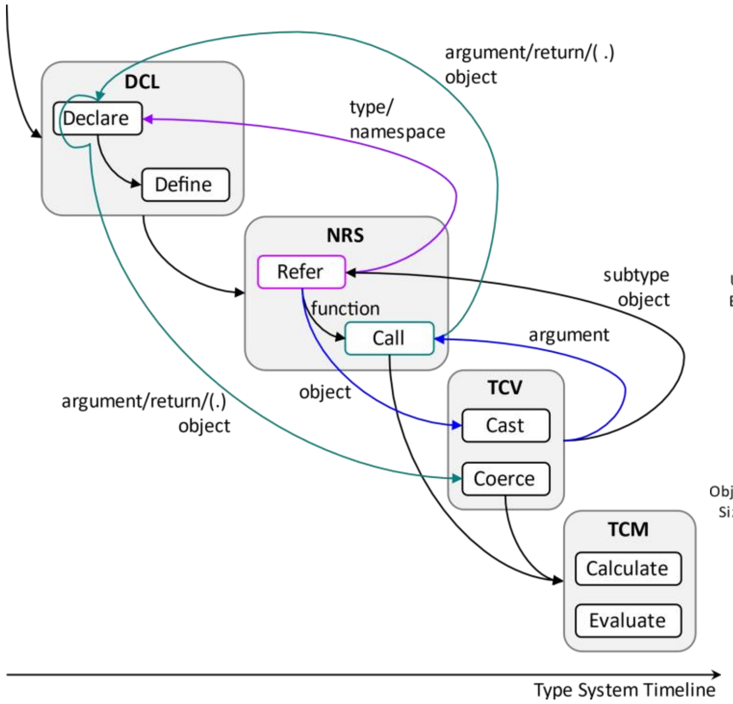
BF Bugs Models

BF Bugs Models

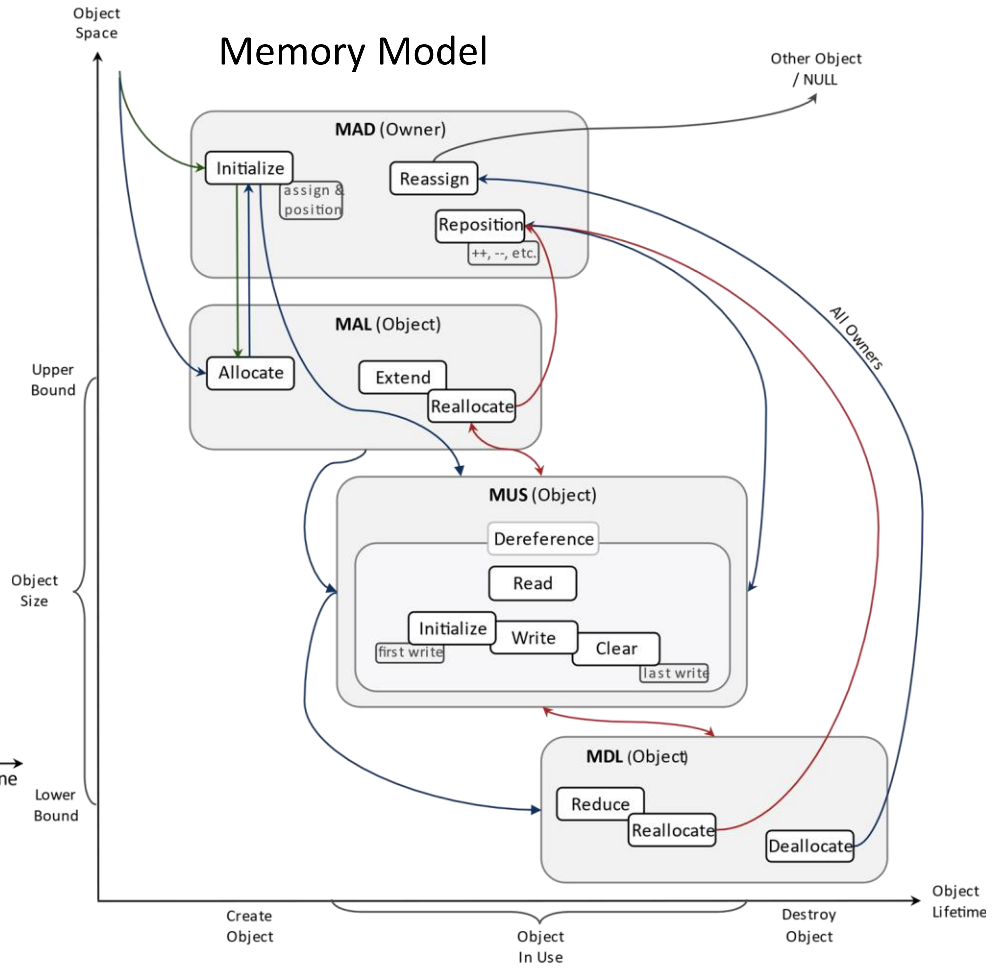
BF Input/Output Model



BF Data Type Model



Memory Model

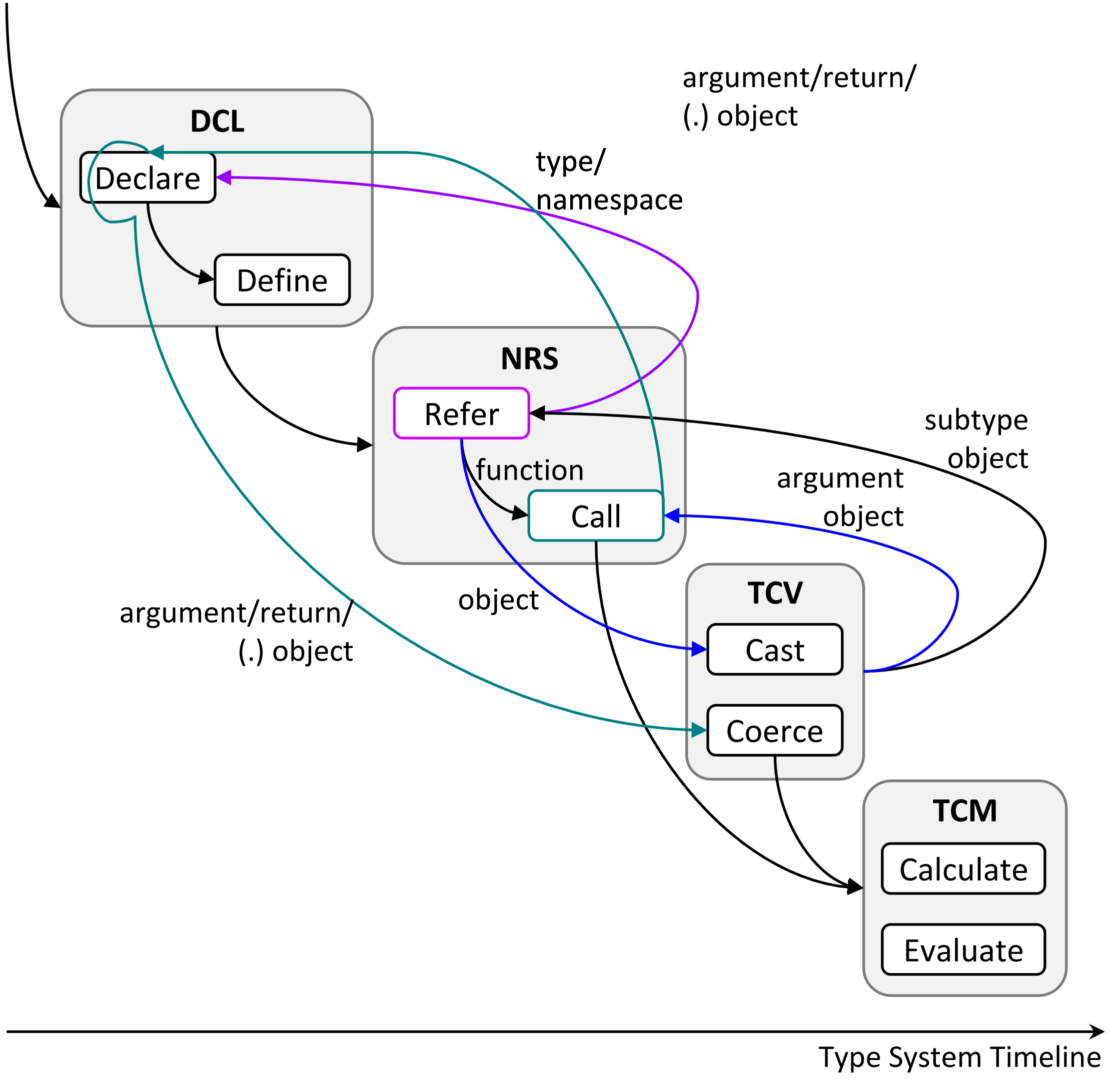


- Identify Secure Code Principles:
  - ○ Input/Output Safety
  - ○ Data Type Safety
  - ○ Memory Safety

Data Type Bugs Model

- Four phases, corresponding to the BF _DAT classes:
  DCL, NRS, TCV, and TCM

- Data Type operations flow



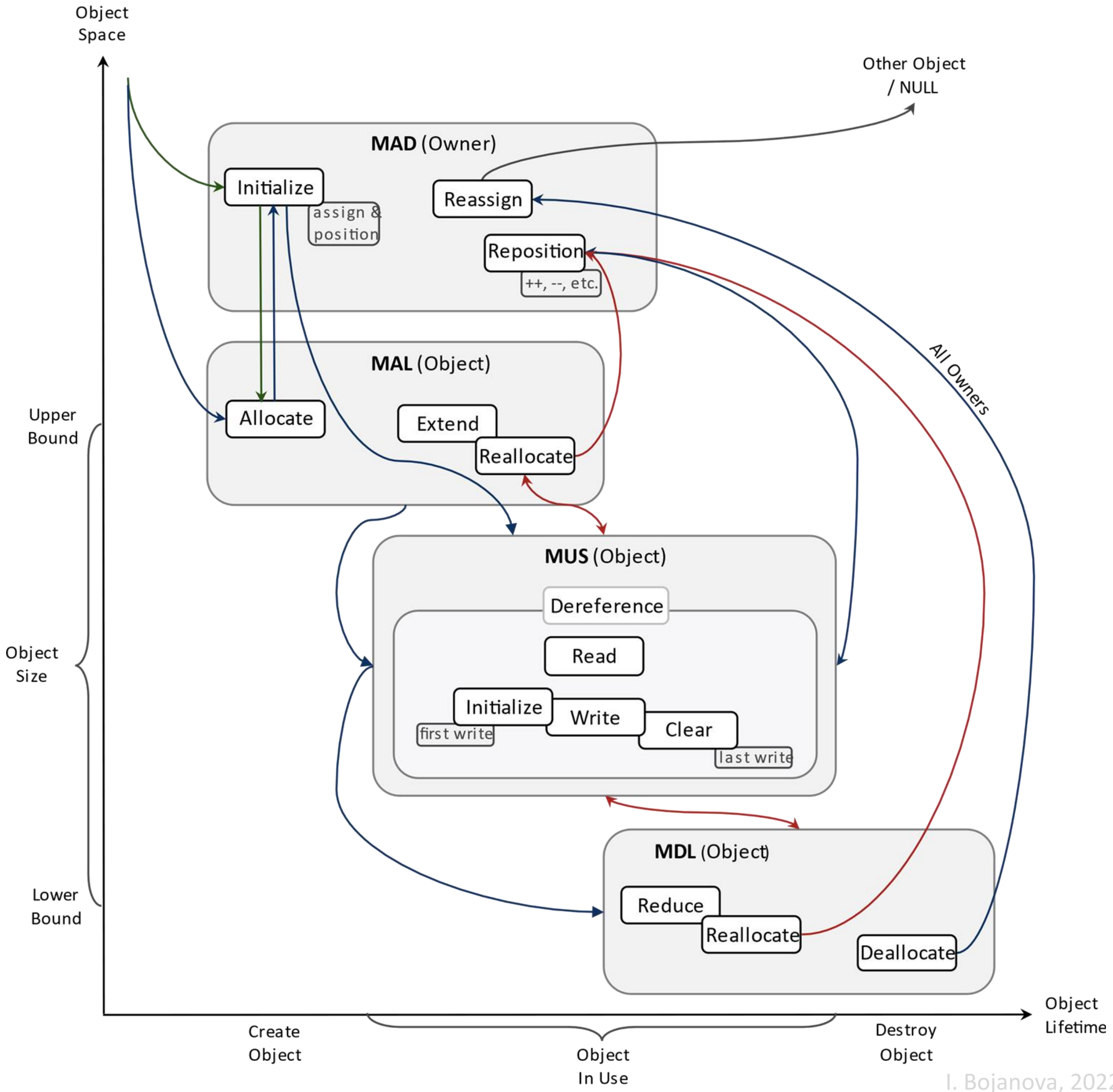
➢ **Entity**:
  - ○ Object
  - ○ Function
  - ○ Data Type
  - ○ Namespace

I. Bojanova, 2022

BF Memory Bugs Model

- Four phases, corresponding to the BF _MEM classes: MAD, MMN (MAL & MDL), and MUS

- Memory operations flow



I. Bojanova, 2022

BF Weakness Taxonomy

BF – Clusters of Bugs Classes

- Input/Output Bugs:
  DVL, DVR
- Data Type Bugs:
  DCL, NRS, TVC, TCM
- Memory Bugs:
  MAD, MAL, MUS, MD
- Cryptography Bugs:
  ENC, VRF, KMN
- Random Numbers Generation Bugs:
  RND, PRN
- Access Control Bugs
- Control Flow Bugs
- Concurrency Bugs
- …

- BF cluster:
  - Bugs Model
  - Set of Classes
- BF class:
  - Set of Operations
  - Set of Causes
  - Set of Consequences

BF Classes

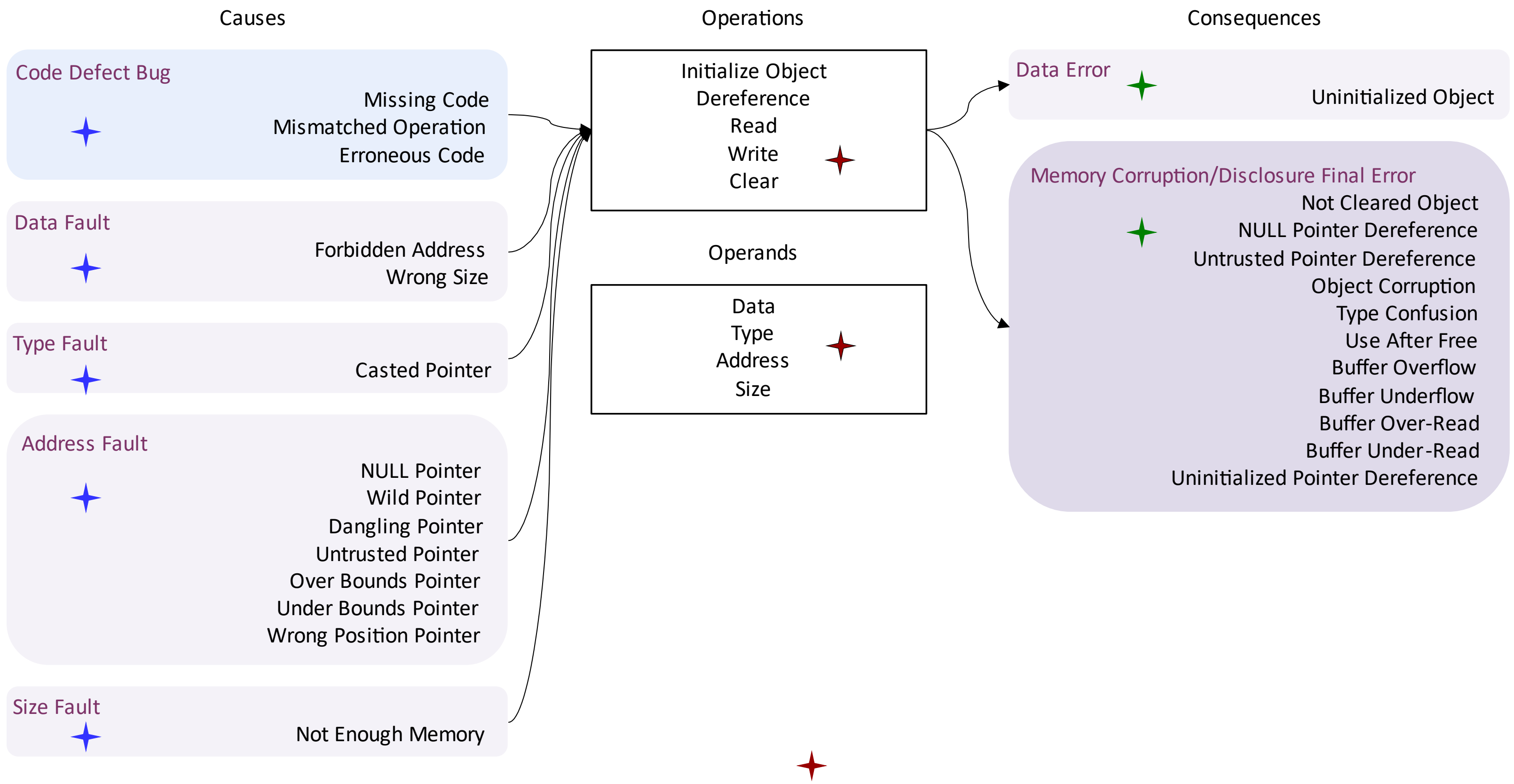
- Structured
- Complete
- Orthogonal
- Language and domain independent

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

BF Classes – Example

Memory Use Bugs (MUS)

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



Causes

Code Defect Bug
Missing Code
Mismatched Operation
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 Free
Buffer Overflow
Buffer Underflow
Buffer Over-Read
Buffer Under-Read
Uninitialized Pointer Dereference

Attributes

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

▨ Bug ▨ Fault/Error ▨ Final Error □ Operation/Operand

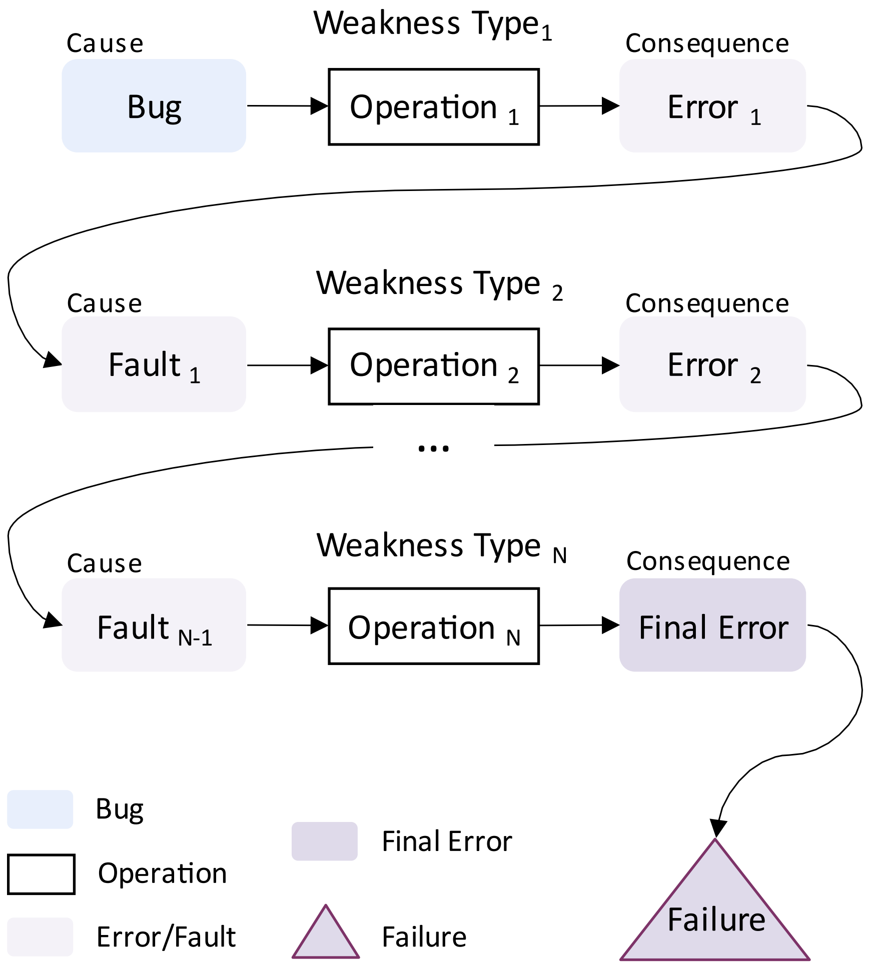
https://samate.nist.gov/BF/
> BF Weakness Taxonomy

I. Bojanova, 2022

BF Vulnerability Model

High Level Vulnerability Model

Bugs Framework (BF) Vulnerability Model



BF, I. Bojanova, 2014-2023

I. Bojanova, 2022

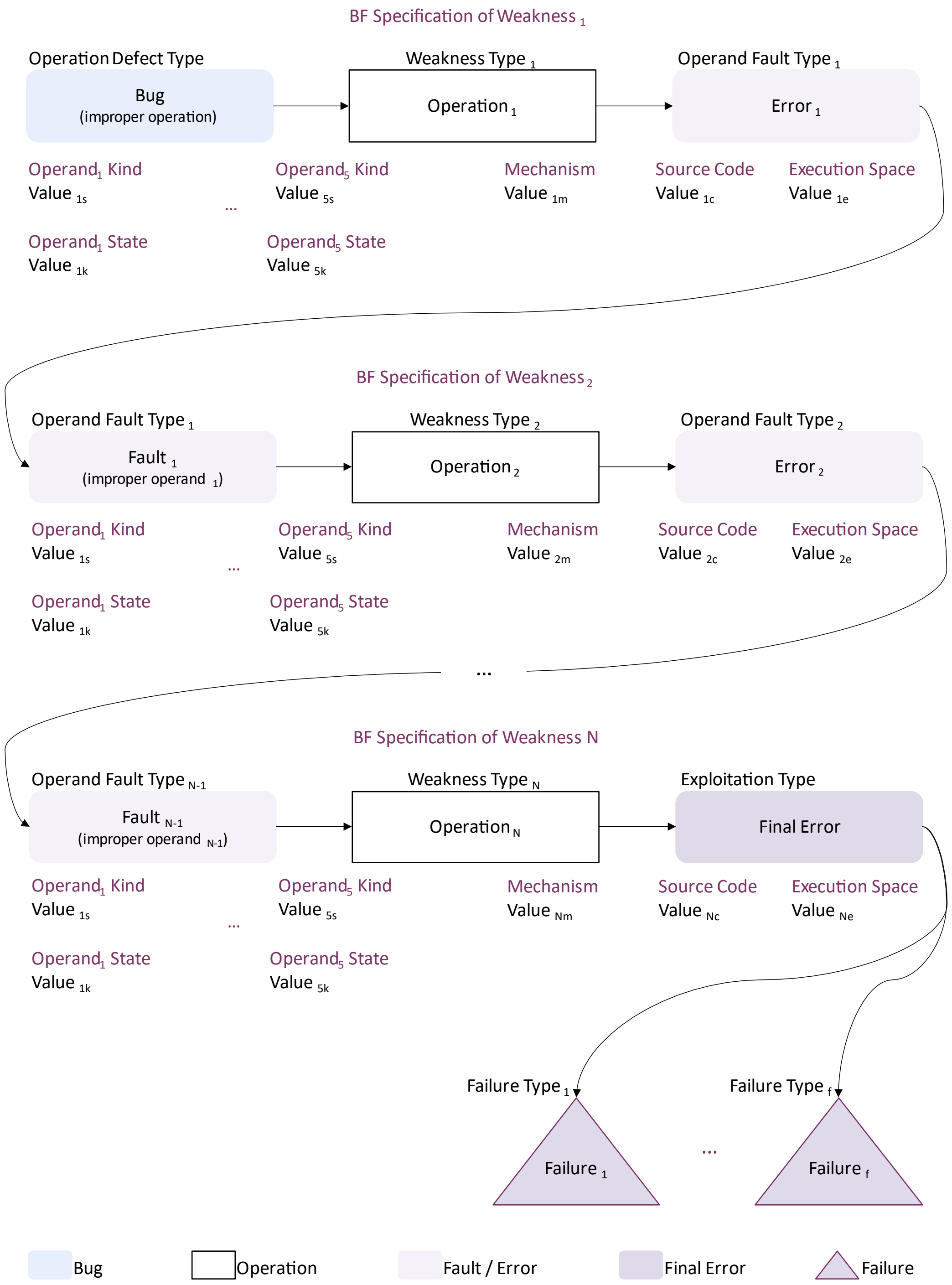
BF Vulnerability Model

A formal BF specification of a chain of underlying BF weakness types, leading to instances of failure types

Operation Defect Types:
Code, Specification

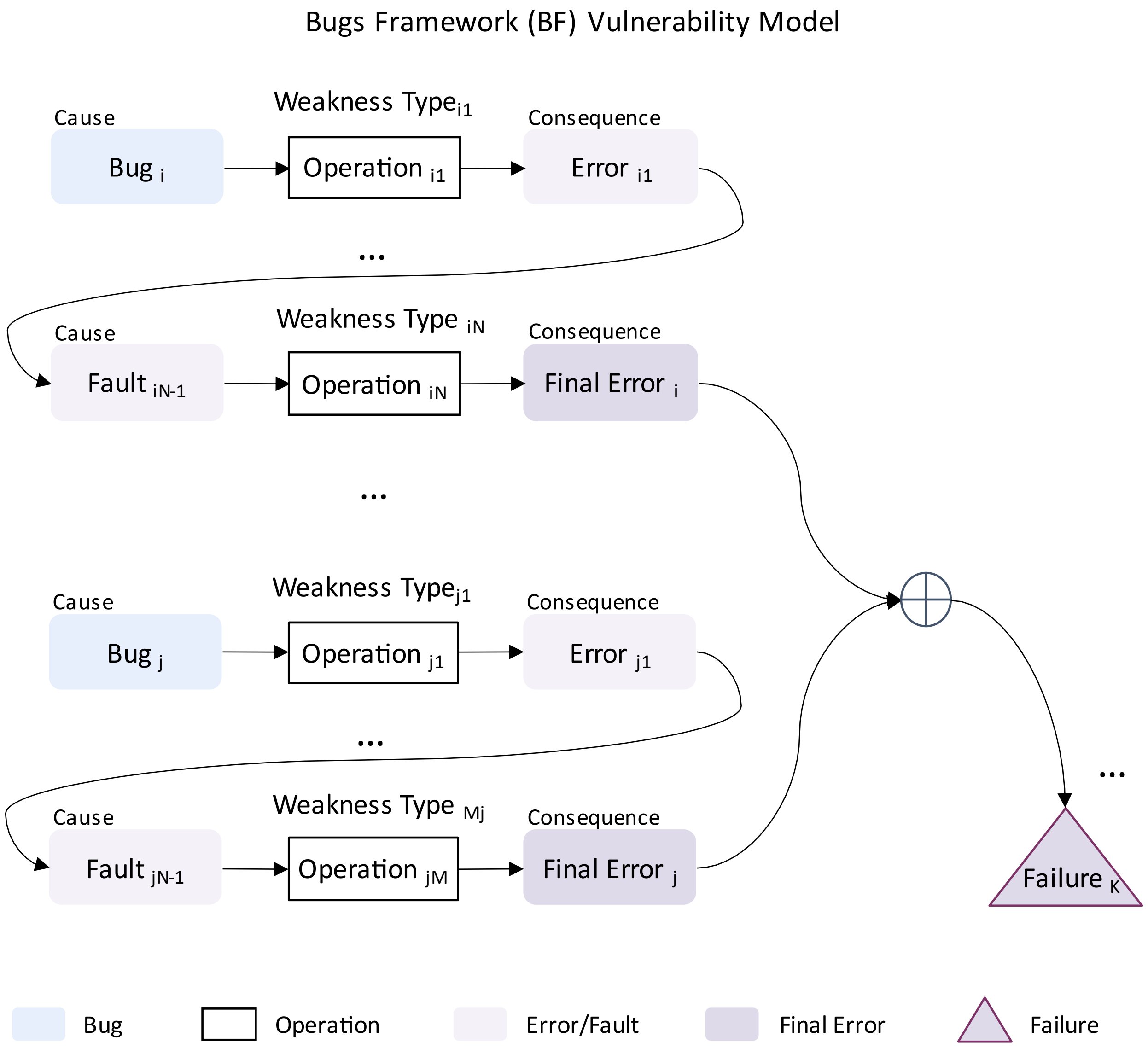
Operand Fault Types:
Name, Data, Type, Address, Size

Bug = improper operation
Fault = improper operand.



BF Model of Converging Vulnerabilities

The bug in at least one of the chains must be fixed to avoid the failure.



BF, I. Bojanova, 2014-2023

BF Formal Language

BF LL(1) Formal Grammar for Class Types _INP, _DAT, _MEM, _CDS

_____________________Variables (Nonterminals)__________________________

V = { Vulnerability, Convergence_Failure, Bug, Operation, OprAttrs_Fault_FinalError, Operation_Attr, Fault, OprdAttrs_Operation, Final_Error, Oprand_Attr, Failure, Code Defect, Specification Defect, DVL_Operation, DVR_Operation, DCL_Operation, NRS_Operation, TCV_Operation, TCM_Operation, MAD_Operation, MMN_Operation, MUS_Operation, KMN_Operation, CPH_Operation, DSV_Operation, Mechanism, Source Code, Execution Space, Data_Fault, Type_Fault, Name_Fault, Address_Fault, Size_Fault, Injection, Access, Type Compute, Memory Corruption/Disclosure, Data Security, Data_Kind, Name_Kind, Type_Kind, Address_Kind, Size_Kind, Data_State, Name_State, Type_State, Address_State, Size_State }

_____________________Tokens (Terminals)__________________________

Σ = {'Missing Code', 'Erroneous Code', 'Under-Restrictive Policy', 'Over-Restrictive Policy', 'Wrong Code', 'Missing Modifier', 'Wrong Modifier', 'Anonymous Scope', 'Wrong Scope', 'Missing Qualifier', 'Wrong Qualifier', 'Mismatched Operation', 'Added Code', 'Wrong Algorithm', 'Hardcoded Key', 'Weak Protocol', 'Validate', 'Sanitize', 'Verify', 'Correct', 'Declare', 'Define', 'Refer', 'Call', 'Cast', 'Coerce', 'Calculate', 'Evaluate', 'Initialize Pointer', 'Reposition', 'Reassign', 'Allocate', 'Extend', 'Reallocate-Extend', 'Deallocate', 'Reduce', 'Reallocate-Reduce', 'Initialize Object', 'Dereference', 'Read', 'Write', 'Clear', 'Generate/Select', 'Store', 'Distribute', 'Use', 'Destroy', 'Encrypt', 'Decrypt', 'Crypto Authenticate', 'Crypto Verify', 'Safelist', 'Denylist', 'Format', 'Length', 'Codebase', 'Third-Party', 'Standard Library', 'Compiler/Interpreter', 'Local', 'Admin', 'Bare-Metal', 'Value', 'Quantity', 'Range', 'Data Type', 'Other Rules', 'Simple', 'Generics', 'Overriding', 'Overloading', 'Resolve', 'Bind', 'Early Bind', 'Late Bind', 'Ad-hoc Bind', 'Pass In', 'Pass Out', 'Function', 'Operator', 'Method', 'Lambda Expression', 'Procedure', 'Direct', 'Sequential', 'Userland', 'Kernel', 'Implicit', 'Explicit', 'Hash + RND', 'MAC', 'Digital Signature', 'Symmetric Algorithm', 'Asymmetric Algorithm', 'Corrupted Data', 'Tampered Data', 'Corrupted Policy Data', 'Tampered Policy Data', 'Invalid Data', 'Wrong Type Resolved', 'Missing Overridden Function', 'Missing Overloaded Function', 'Incomplete Type', 'Wrong Generic Type', 'Confused Subtype', 'Wrong Argument Type', 'Wrong Object Resolved', 'Wrong Object Type Resolved', 'Under Range', 'Over Range', 'Flipped Sign', 'Wrong Type', 'Mismatched Argument', 'Wrong Function Resolved', 'Wrong Generic Function Bound', 'Wrong Overridden Function Bound', 'Wrong Overloaded Function Bound', 'Wrong Argument', 'Reference vs. Object', 'Hardcoded Address', 'Single Owned Address', 'Wrong Index', 'Wrong Size', 'Wrong Index Type', 'Casted Pointer', 'NULL Pointer', 'Wild Pointer', 'Dangling Pointer', 'Untrusted Pointer', 'Over Bounds Pointer', 'Under Bounds Pointer', 'Wrong Position Pointer', 'Not Enough Memory', 'Forbidden Address', 'Weak Keying Material', 'Weak Ciphertext', 'Unverified Data', 'Weak Key', 'Weak Random Bits', 'Repeated IV', 'Weak Shared Secrets', 'Revealed Key', 'Entered', 'Stored', 'In Use', 'Transferred', 'Object', 'Function', 'Data Type', 'Namespace', 'Primitive', 'Structure', 'Resolved', 'Bound', 'Numeric', 'Text', 'Pointer', 'Boolean', 'Stack', 'Heap', '/other/', 'Actual', 'Used', 'Huge', 'Moderate', 'Little', 'Hashes', 'Keying Material', 'Digital Certificate', 'Credentials', 'System Data', 'State Data', 'Cryptographic', 'Digital Document', 'Secret', 'Private', 'Public', 'Query Injection', 'Command Injection', 'Source Code Injection', 'Parameter Injection', 'File Injection', 'Wrong Access Object', 'Wrong Access Type', 'Wrong Access Function', 'Undefined', 'Memory Leak', 'Memory Overflow', 'Double Free', 'Object Corruption', 'Not Cleared Object', 'NULL Pointer Dereference', 'Untrusted Pointer Dereference', 'Type Confusion', 'Use After Free', 'Buffer Overflow', 'Buffer Underflow', 'Buffer Over-Read', 'Buffer Under-Read', 'Uninitialized Pointer Dereference', 'Revealed IV', 'Revealed Shared Secrets', 'Revealed Domain Parameter', 'Revealed Random Bits', 'Revealed Plaintext', 'Revealed Key', 'Forged Signature', 'Spoofed Identity', '⨁' }

BF LL(1) Formal Grammar

_____________________Rules (Productions)__________________________

S → Vulnerability Convergence_Failure

Vulnerability → Bug Operation OprAttrs_Fault_FinalError

OprAttrs_Fault_FinalError → Operation_Attr OprAttrs_Fault_FinalError | Fault OprdAttrs_Operation | Final_Error

OprdAttrs_Operation → Oprand_Attr OprdAttrs_Operation | Operation OprAttrs_Fault_FinalError

Convergence_Failure → '⨁' Vulnerability Convergence_Failure | **Failure** ε

- -

Bug → Code_Defect | Specification_Defect

Code_Defect → 'Missing Code' | 'Erroneous Code' | 'Wrong Code' | 'Mismatched Operation' | 'Added Code'

Specification_Defect → 'Under-Restrictive Policy' | 'Over-Restrictive Policy' | 'Missing Modifier' | 'Wrong Modifier' | 'Anonymous Scope' | 'Wrong Scope' | 'Missing Qualifier' | 'Wrong Qualifier' | 'Wrong Algorithm' | 'Hardcoded Key' | 'Weak Protocol'

- -

Operation → DVL_Operation | DVR_Operation | DCL_Operation | NRS_Operation | TCV_Operation | TCM_Operation | MAD_Operation | MMN_Operation | MUS_Operation | KMN_Operation | CPH_Operation | DSV_Operation

DVL_Operation → 'Validate' | 'Sanitize'

DVR_Operation → 'Verify' | 'Correct'

DCL_Operation → 'Declare' | 'Define'

NRS_Operation → 'Refer' | 'Call'

TCV_Operation → 'Cast' | 'Coerce'

TCM_Operation → 'Calculate' | 'Evaluate'

MAD_Operation → 'Initialize Pointer' | 'Reposition' | 'Reassign'

MMN_Operation → 'Allocate' | 'Extend' | 'Reallocate-Extend' | 'Deallocate' | 'Reduce' | 'Reallocate-Reduce'

MUS_Operation → 'Initialize Object' | 'Dereference' | 'Read' | 'Write' | 'Clear'

KMN_Operation → 'Generate/Select' | 'Store' | 'Distribute' | 'Use' | 'Destroy'

CPH_Operation → 'Encrypt' | 'Decrypt'

DSV_Operation → 'Crypto Authenticate' | 'Crypto Verify'

Operand → 'Data' | 'Name' | 'Type' | 'Address' | 'Size'

BF LL(1) Formal Grammar

Fault → Data_Fault | Type_Fault | Name_Fault | Address_Fault | Size_Fault
Data_Fault → 'Corrupted Data' | 'Tampered Data' | 'Corrupted Policy Data' | 'Tampered Policy Data' | 'Invalid Data' | 'Under Range' | 'Over Range' | 'Flipped Sign' | 'Wrong Argument' | 'Reference vs. Object' | 'Hardcoded Address' | 'Single Owned Address' | 'Wrong Index' | 'Wrong Size' | 'Forbidden Address' | 'Weak Keying Material' | 'Weak Ciphertext' | 'Unverified Data' | 'Weak Key' | 'Weak Random Bits' | 'Repeated IV' | 'Weak Shared Secrets' | 'Revealed Key'
Type_Fault → 'Wrong Type Resolved' | 'Incomplete Type' | 'Wrong Generic Type' | 'Confused Subtype' | 'Wrong Argument Type' | 'Wrong Type' | 'Mismatched Argument' | 'Wrong Object Type Resolved' | 'Wrong Index Type' | 'Casted Pointer'
Name_Fault → 'Missing Overridden Function' | 'Missing Overloaded Function' | 'Wrong Object Resolved' | 'Wrong Object Type Resolved' | 'Wrong Function Resolved' | 'Wrong Generic Function Bound' | 'Wrong Overridden Function Bound' | 'Wrong Overloaded Function Bound'
Address_Fault → 'NULL Pointer' | 'Wild Pointer' | 'Dangling Pointer' | 'Untrusted Pointer' | 'Over Bounds Pointer' | 'Under Bounds Pointer' | 'Wrong Position Pointer'
Size_Fault → 'Not Enough Memory'
- -
Final_Error → Injection | Access | Type_Compute | Memory_Corruption/Disclosure | Data_Security
Injection → 'Query Injection' | 'Command Injection' | 'Source Code Injection' | 'Parameter Injection' | 'File Injection'
Access → 'Wrong Access Object' | 'Wrong Access Type' | 'Wrong Access Function'
Type_Compute → 'Undefined'
Memory_Corruption/Disclosure → 'Memory Leak' | 'Memory Overflow' | 'Double Free' | 'Object Corruption' | 'Not Cleared Object' | 'NULL Pointer Dereference' | 'Untrusted Pointer Dereference' | 'Type Confusion' | 'Use After Free' | 'Buffer Overflow' | 'Buffer Underflow' | 'Buffer Over-Read' | 'Buffer Under-Read' | 'Uninitialized Pointer Dereference'
Data_Security → 'Revealed IV' | 'Revealed Shared Secrets' | 'Revealed Domain Parameter' | 'Revealed Random Bits' | 'Revealed Plaintext' | 'Revealed Key' | 'Forged Signature' | 'Spoofed Identity'

BF LL(1) Formal Grammar

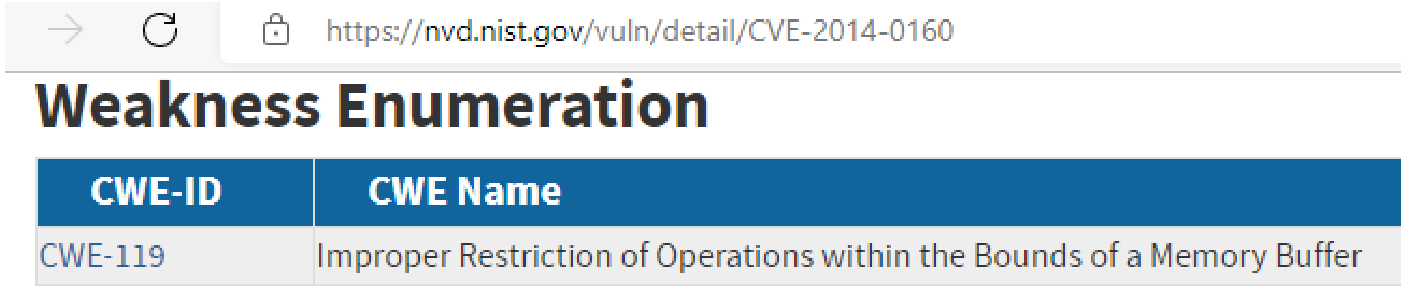
Operation_Attr → Mechanism | Source_Code | Execution_Space
Mechanism → 'Safelist' | 'Denylist' | 'Format' | 'Length' | 'Value' | 'Quantity' | 'Range' | 'Data Type' | 'Other Rules' | 'Simple' | 'Generics' | 'Overriding' | 'Overloading' | 'Resolve' | 'Bind' | 'Early Bind' | 'Late Bind' | 'Ad-hoc Bind' | 'Pass In' | 'Pass Out' | 'Function' | 'Operator' | 'Method' | 'Lambda Expression' | 'Procedure' | 'Direct' | 'Sequential' | 'Implicit' | 'Explicit' | 'Hash + RND' | 'MAC' | 'Digital Signature' | 'Symmetric Algorithm' | 'Asymmetric Algorithm'
Source_Code → 'Codebase' | 'Third-Party' | 'Standard Library' | 'Compiler/Interpreter'
Execution_Space → 'Local' | 'Admin' | 'Bare-Metal' | 'Userland' | 'Kernel'
- -
Oprand_Attr → Data_Kind | Name_Kind | Type_Kind | Address_Kind | Size_Kind | Data_State | Name_State | Type_State | Address_State | Size_State
Data_State → 'Entered' | 'Stored' | 'In Use' | 'Transferred'
Data_Kind → 'Numeric' | 'Text' | 'Pointer' | 'Boolean' | 'Hashes' | 'Keying Material' | 'Digital Certificate' | 'Credentials' | 'System Data' | 'State Data' | 'Cryptographic' | 'Digital Document' | 'Secret' | 'Private' | 'Public'
Name_Kind → 'Object' | 'Function' | 'Data Type' | 'Namespace'
Name_State → 'Resolved' | 'Bound'
Type_Kind → 'Primitive' | 'Structure'
Address_State → 'Stack' | 'Heap' | '/other/'
Address_Kind → 'Huge' | 'Moderate' | 'Little'
Size_Kind → 'Actual' | 'Used'

BF Specification 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.



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)

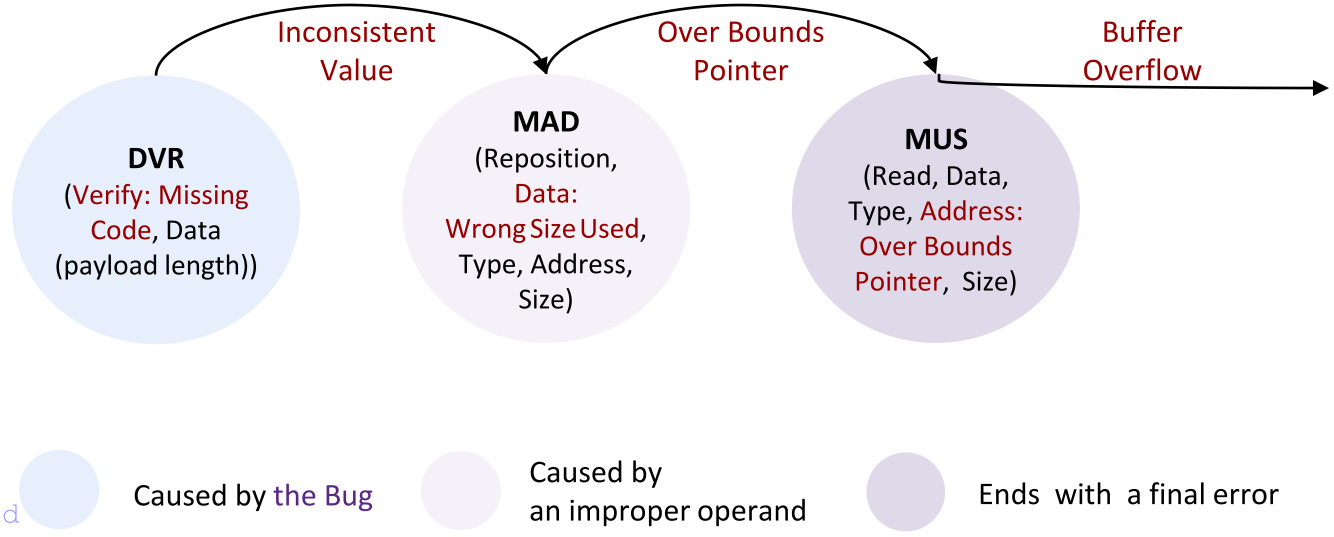
CVE-2014-0160 The (1) TLS and (2) DTLS implementations in OpenSSL 1.0.1 before 1.0.1g do not properly handle Heartbeat Extension packets, which allows remote attackers to obtain sensitive information from process memory via crafted packets that trigger a buffer over-read, as demonstrated by reading private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

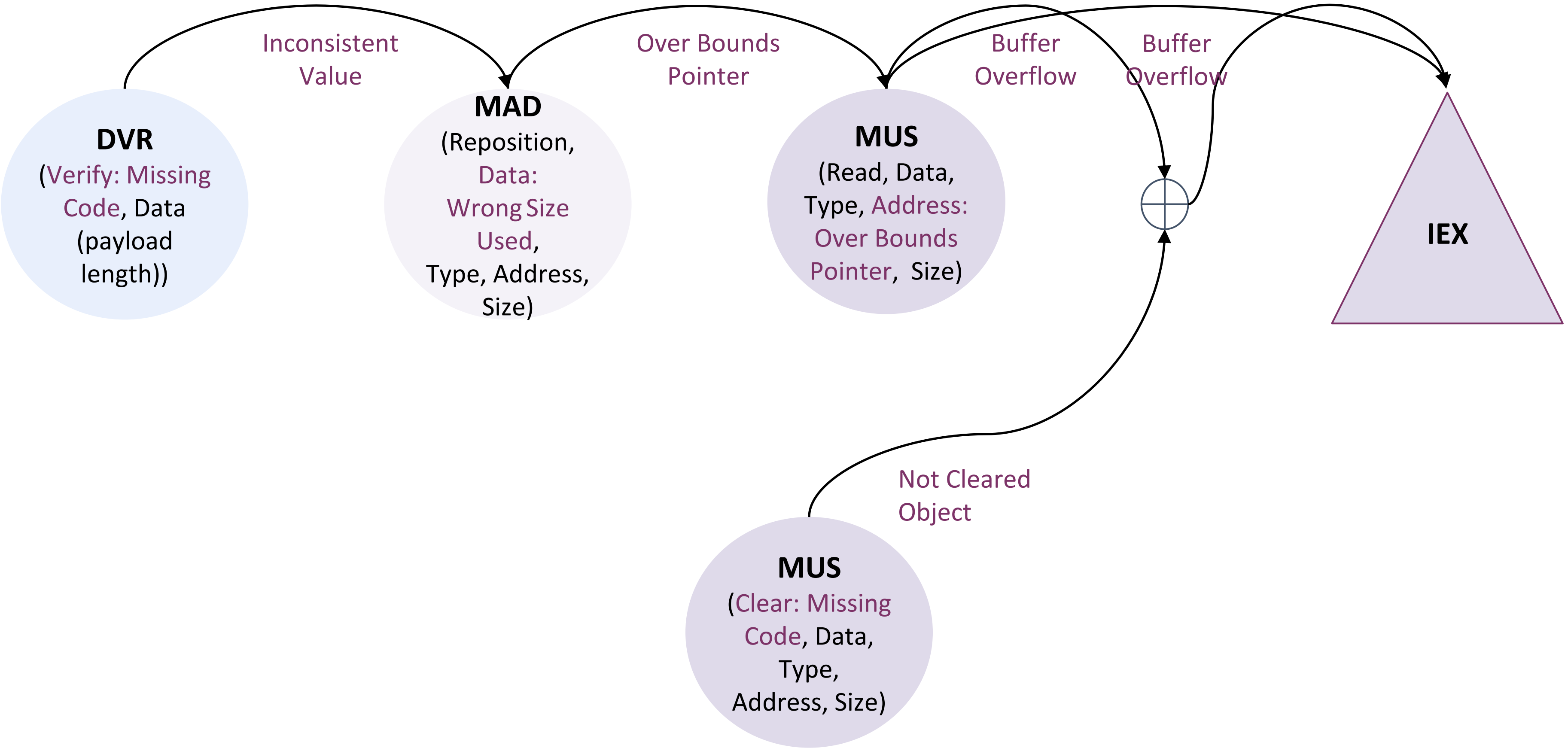
```
/* Naive implementation of memcpy
void *memcpy (void *dst, const void *src, size_t n)
{
    size_t i;     payload
    for (i=0; i<n; i++)
        *(char *) dst++ = *(char *) src++;
    return dst;        bp                pl
}
```



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, 2022

Heartbleed (CVE-2014-0160)



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

Inconsistent Value

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

Over Bounds Pointer

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

Buffer Overflow

Buffer Overflow

IEX

Not Cleared Object

MUS (Clear: Missing Code, Data, Type, Address, Size)

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

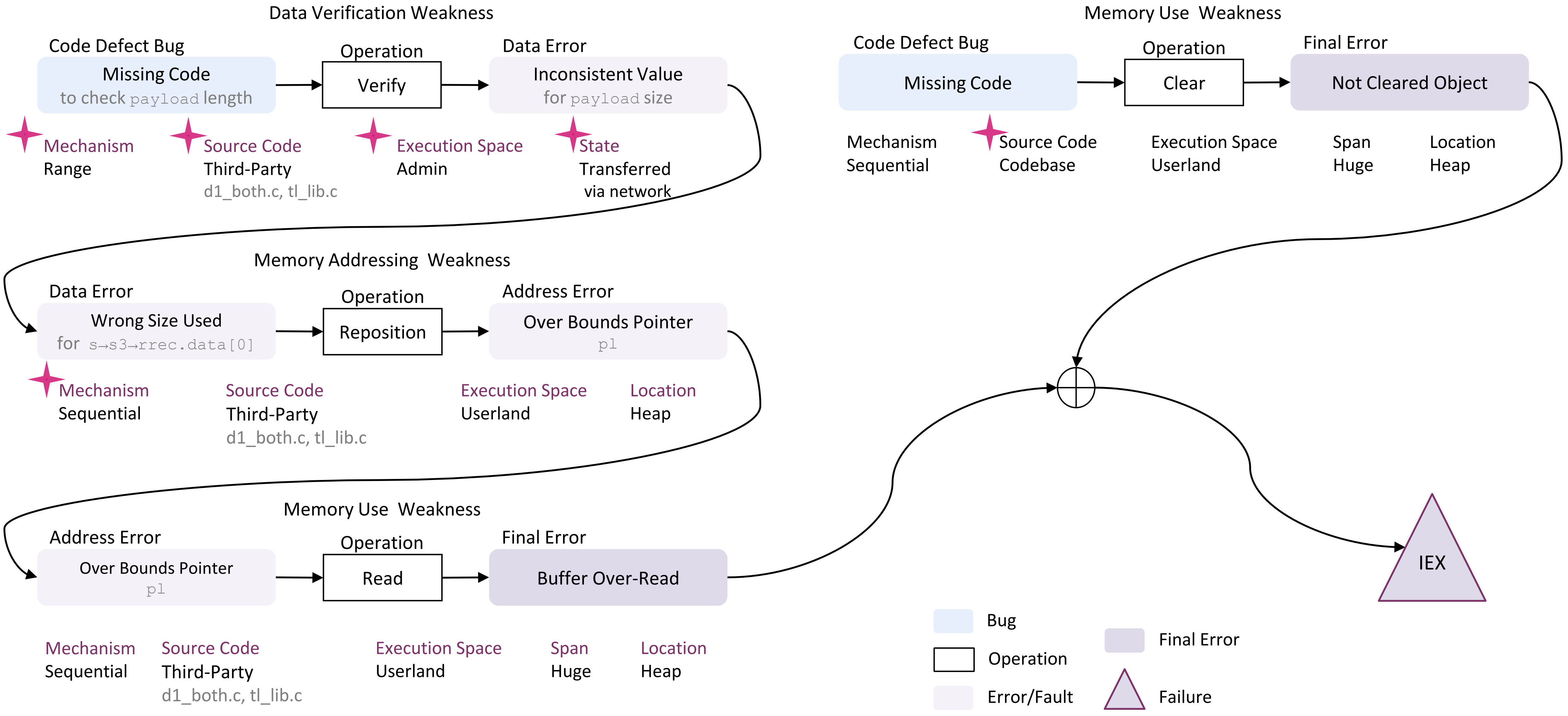
● Intermediate State – caused by at least one operand is improper

● Final State – ends with a final error

△ Failure – caused by a final error

I. Bojanova, 2022

Heartbleed



Data Verification Weakness

Code Defect Bug: Missing Code to check payload length → Operation: Verify → Data Error: Inconsistent Value for payload size

Mechanism Range | Source Code Third-Party d1_both.c, tl_lib.c | Execution Space Admin | State Transferred via network

Memory Addressing Weakness

Data Error: Wrong Size Used for s→s3→rrec.data[0] → Operation: Reposition → Address Error: Over Bounds Pointer pl

Mechanism Sequential | Source Code Third-Party d1_both.c, tl_lib.c | Execution Space Userland | Location Heap

Memory Use Weakness

Address Error: Over Bounds Pointer pl → Operation: Read → Final Error: Buffer Over-Read

Mechanism Sequential | Source Code Third-Party d1_both.c, tl_lib.c | Execution Space Userland | Span Huge | Location Heap

Memory Use Weakness

Code Defect Bug: Missing Code → Operation: Clear → Final Error: Not Cleared Object

Mechanism Sequential | Source Code Codebase | Execution Space Userland | Span Huge | Location Heap

IEX

Bug

Operation

Error/Fault

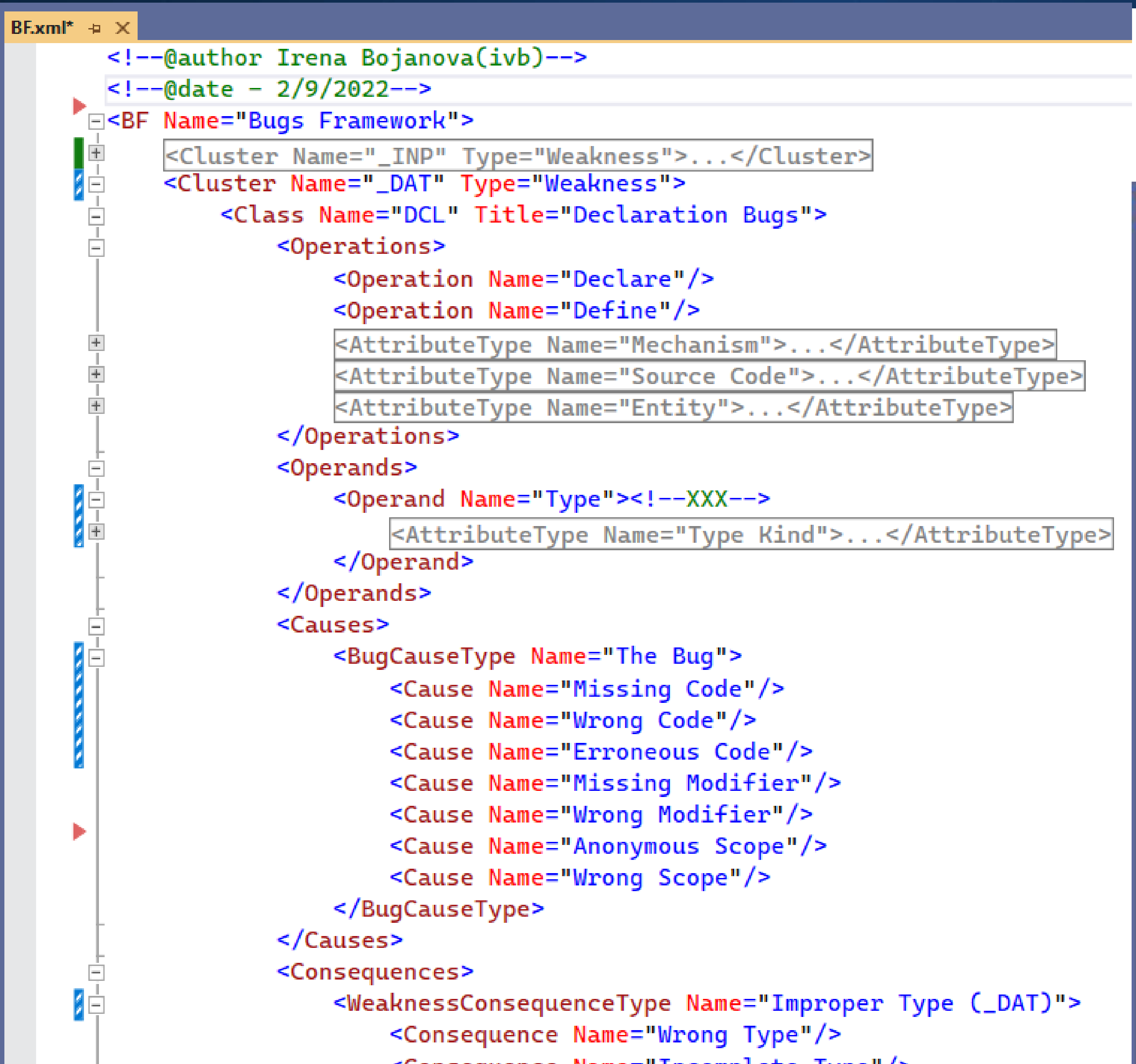
Final Error

Failure

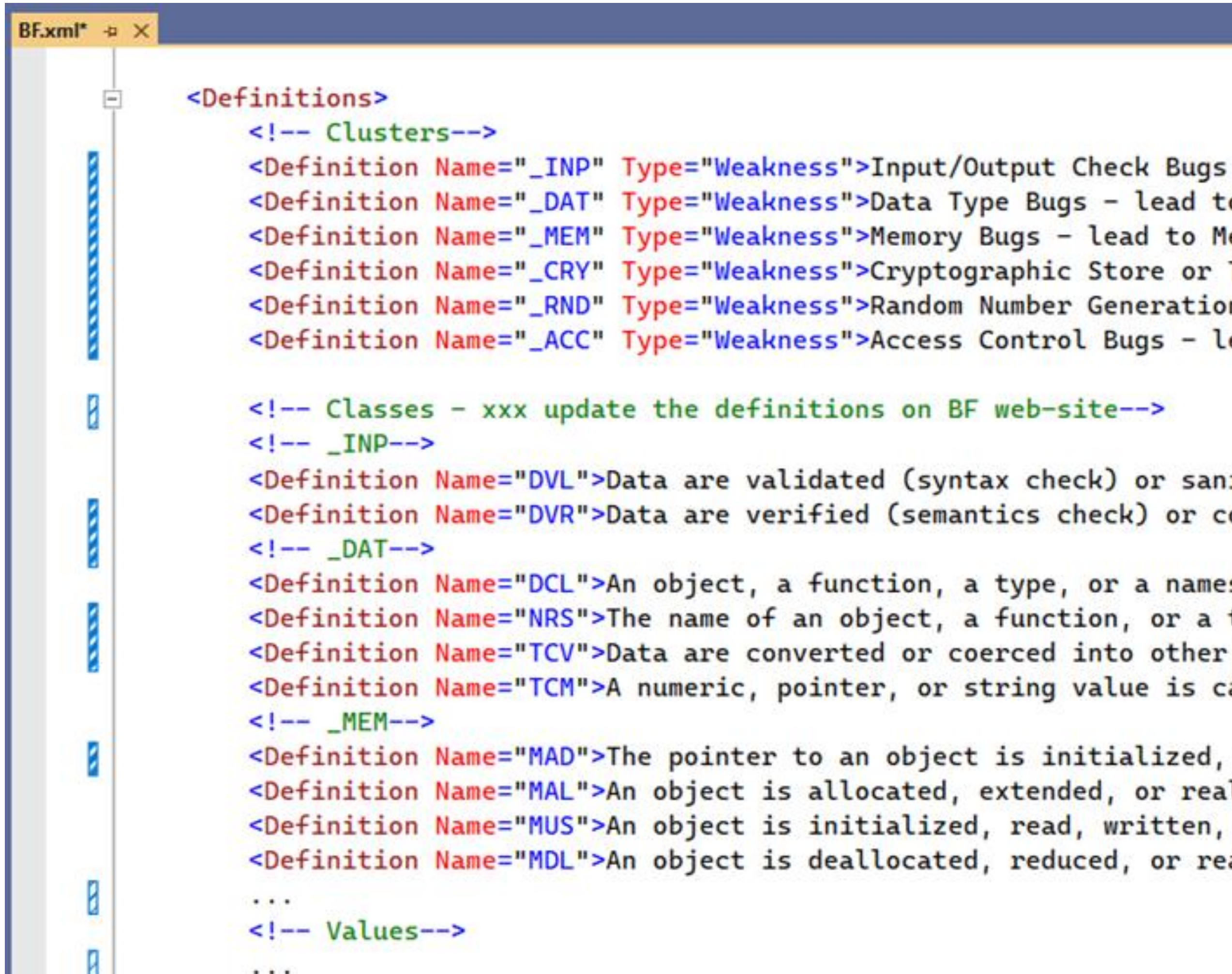
I. Bojanova, 2022

BF Generation Tools

BF Taxonomy – BF.xml



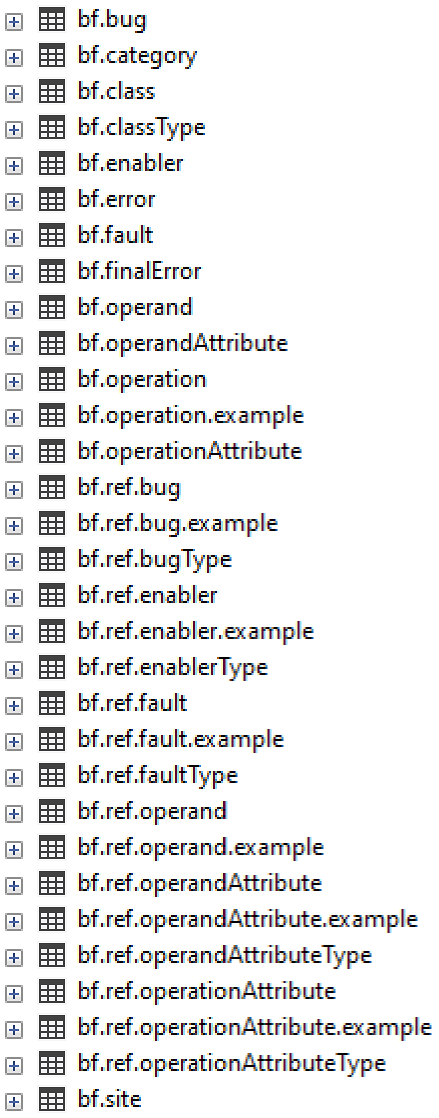
```
<!--@author Irena Bojanova(ivb)-->
<!--@date - 2/9/2022-->
<BF Name="Bugs Framework">
    <Cluster Name="_INP" Type="Weakness">...</Cluster>
    <Cluster Name="_DAT" Type="Weakness">
        <Class Name="DCL" Title="Declaration Bugs">
            <Operations>
                <Operation Name="Declare"/>
                <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"/>
                    <Consequence Name="Incomplete Type"/>
```



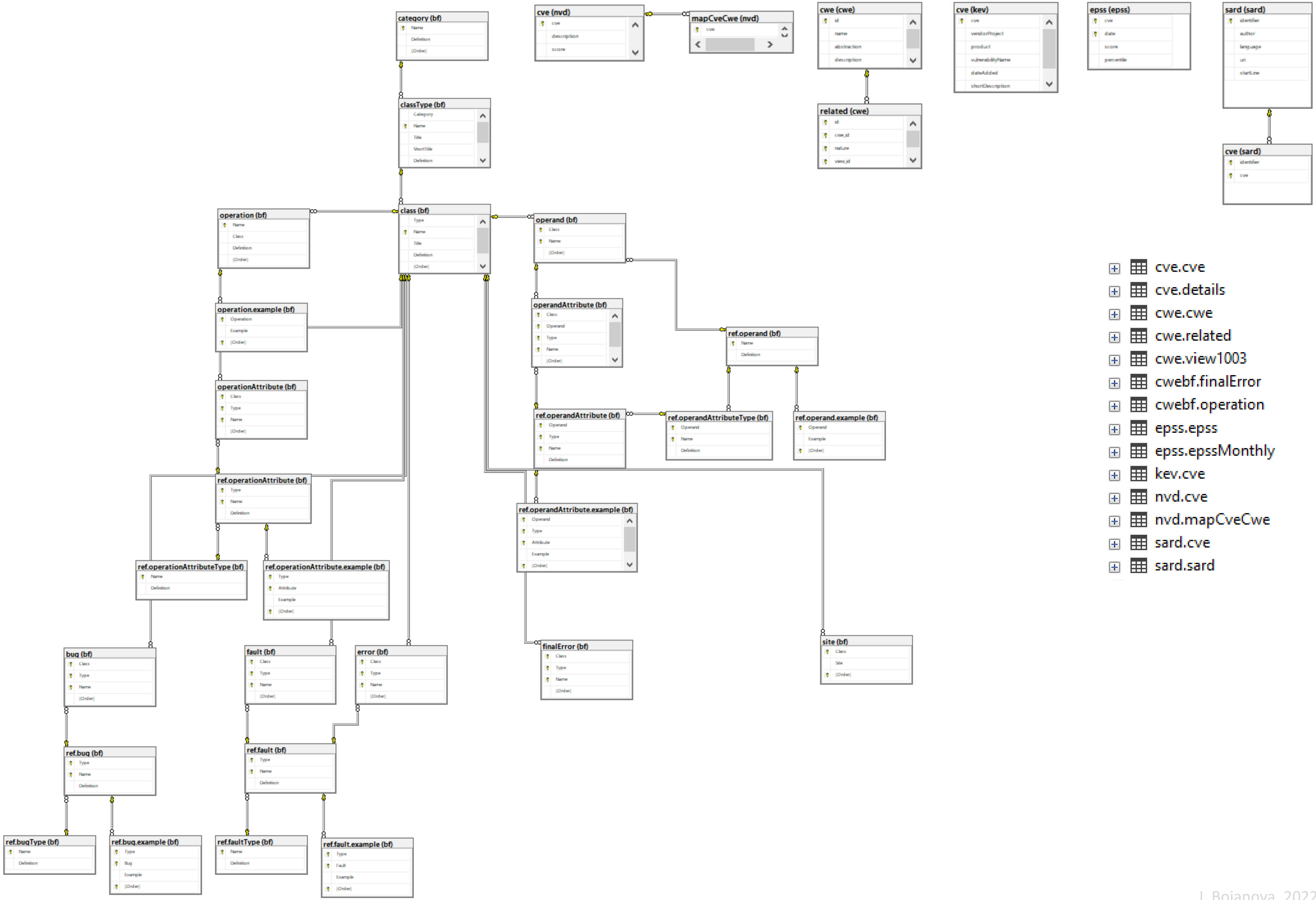
```
<Definitions>
    <!-- Clusters-->
    <Definition Name="_INP" Type="Weakness">Input/Output Check Bugs
    <Definition Name="_DAT" Type="Weakness">Data Type Bugs - lead t
    <Definition Name="_MEM" Type="Weakness">Memory Bugs - lead to M
    <Definition Name="_CRY" Type="Weakness">Cryptographic 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 DB



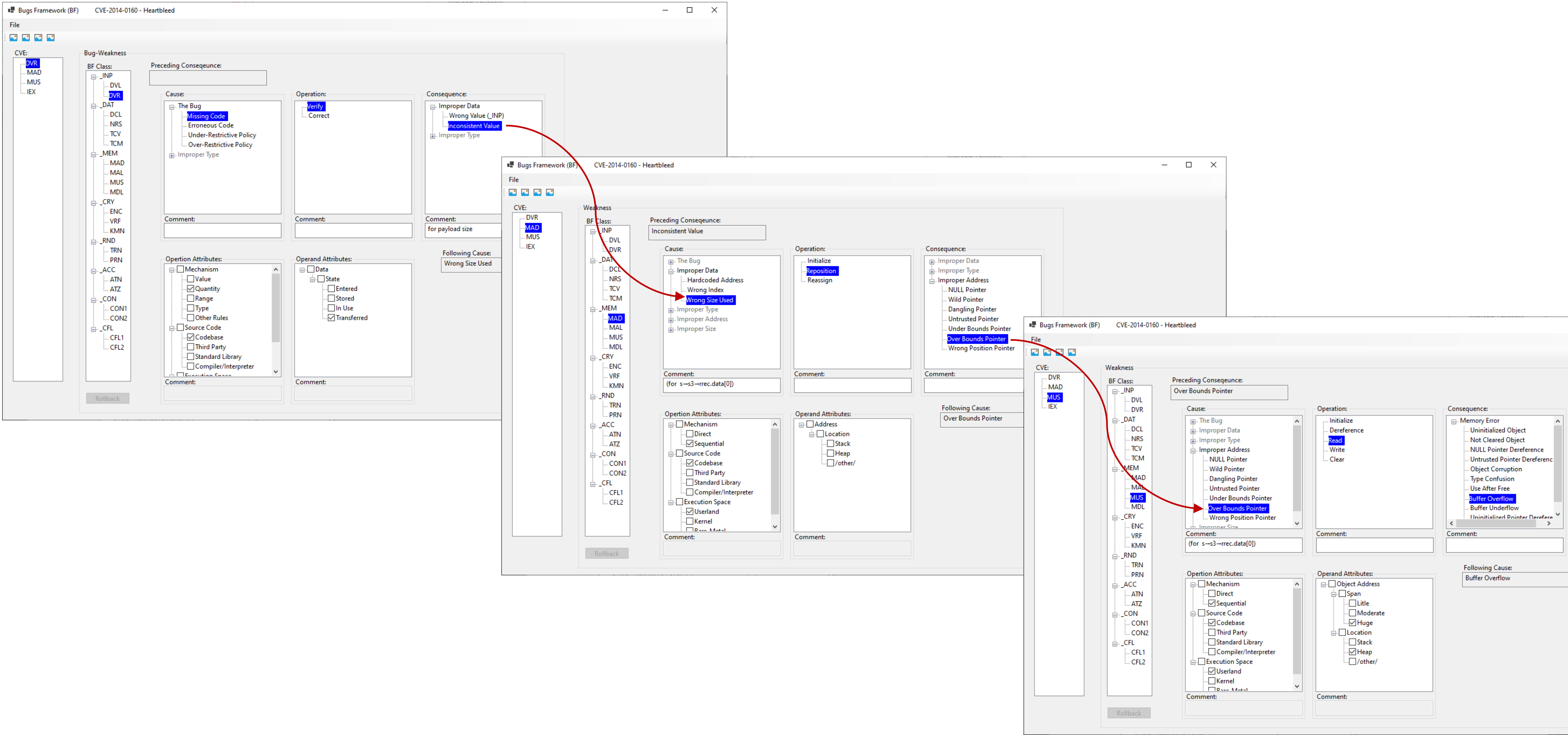
- bf.bug
- bf.category
- bf.class
- bf.classType
- bf.enabler
- bf.error
- bf.fault
- bf.finalError
- bf.operand
- bf.operandAttribute
- bf.operation
- bf.operation.example
- bf.operationAttribute
- bf.ref.bug
- bf.ref.bug.example
- bf.ref.bugType
- bf.ref.enabler
- bf.ref.enabler.example
- bf.ref.enablerType
- bf.ref.fault
- bf.ref.fault.example
- bf.ref.faultType
- bf.ref.operand
- bf.ref.operand.example
- bf.ref.operandAttribute
- bf.ref.operandAttribute.example
- bf.ref.operandAttributeType
- bf.ref.operationAttribute
- bf.ref.operationAttribute.example
- bf.ref.operationAttributeType
- bf.site



- cve.cve
- cve.details
- cwe.cwe
- cwe.related
- cwe.view1003
- cwebf.finalError
- cwebf.operation
- epss.epss
- epss.epssMonthly
- kev.cve
- nvd.cve
- nvd.mapCveCwe
- sard.cve
- sard.sard

CVE-2014-0160 - Heartbleed.bfcve

NIST



Bugs Framework (BF) CVE-2014-0160 - Heartbleed

CVE: DVR, MAD, MUS, IEX

Bug-Weakness

BF Class: _INP (DVL, DVR), _DAT (DCL, NRS, TCV, TCM), _MEM (MAD, MAL, MUS, MDL), _CRY (ENC, VRF, KMN), _RND (TRN, PRN), _ACC (ATN, ATZ), _CON (CON1, CON2), _CFL (CFL1, CFL2)

Preceding Consequence:

Cause: The Bug – **Missing Code**; Erroneous Code; Under-Restrictive Policy; Over-Restrictive Policy; Improper Type

Operation: **Verify**; Correct

Consequence: Improper Data – Wrong Value (_INP); **Inconsistent Value**; Improper Type

Comment: for payload size

Following Cause: Wrong Size Used

Opertion Attributes:
- Mechanism: ☐ Value ☑ Quantity ☐ Range ☐ Type ☐ Other Rules
- Source Code: ☑ Codebase ☐ Third Party ☐ Standard Library ☐ Compiler/Interpreter

Operand Attributes:
- Data – State: ☐ Entered ☐ Stored ☐ In Use ☑ Transferred

Bugs Framework (BF) CVE-2014-0160 - Heartbleed

CVE: DVR, MAD, MUS, IEX

Weakness

BF Class: _INP (DVL, DVR), _DAT (DCL, NRS, TCV, TCM), _MEM (MAD, MAL, MUS, MDL), _CRY (ENC, VRF, KMN), _RND (TRN, PRN), _ACC (ATN, ATZ), _CON (CON1, CON2), _CFL (CFL1, CFL2)

Preceding Consequence: Inconsistent Value

Cause: The Bug; Improper Data – Hardcoded Address; Wrong Index; **Wrong Size Used**; Improper Type; Improper Address; Improper Size

Comment: (for s→s3→rrec.data[0])

Operation: Initialize; **Reposition**; Reassign

Consequence: Improper Data; Improper Type; Improper Address – NULL Pointer; Wild Pointer; Dangling Pointer; Untrusted Pointer; Under Bounds Pointer; **Over Bounds Pointer**; Wrong Position Pointer

Following Cause: Over Bounds Pointer

Opertion Attributes:
- Mechanism: ☐ Direct ☑ Sequential
- Source Code: ☑ Codebase ☐ Third Party ☐ Standard Library ☐ Compiler/Interpreter
- Execution Space: ☑ Userland ☐ Kernel ☐ Bare-Metal

Operand Attributes:
- Address – Location: ☐ Stack ☐ Heap ☐ /other/

Bugs Framework (BF) CVE-2014-0160 - Heartbleed

CVE: DVR, MAD, MUS, IEX

Weakness

BF Class: _INP (DVL, DVR), _DAT (DCL, NRS, TCV, TCM), _MEM (MAD, MAL, MUS, MDL), _CRY (ENC, VRF, KMN), _RND (TRN, PRN), _ACC (ATN, ATZ), _CON (CON1, CON2), _CFL (CFL1, CFL2)

Preceding Consequence: Over Bounds Pointer

Cause: The Bug; Improper Data; Improper Type; Improper Address – NULL Pointer; Wild Pointer; Dangling Pointer; Untrusted Pointer; Under Bounds Pointer; **Over Bounds Pointer**; Wrong Position Pointer; Improper Size

Comment: (for s→s3→rrec.data[0])

Operation: Initialize; Dereference; **Read**; Write; Clear

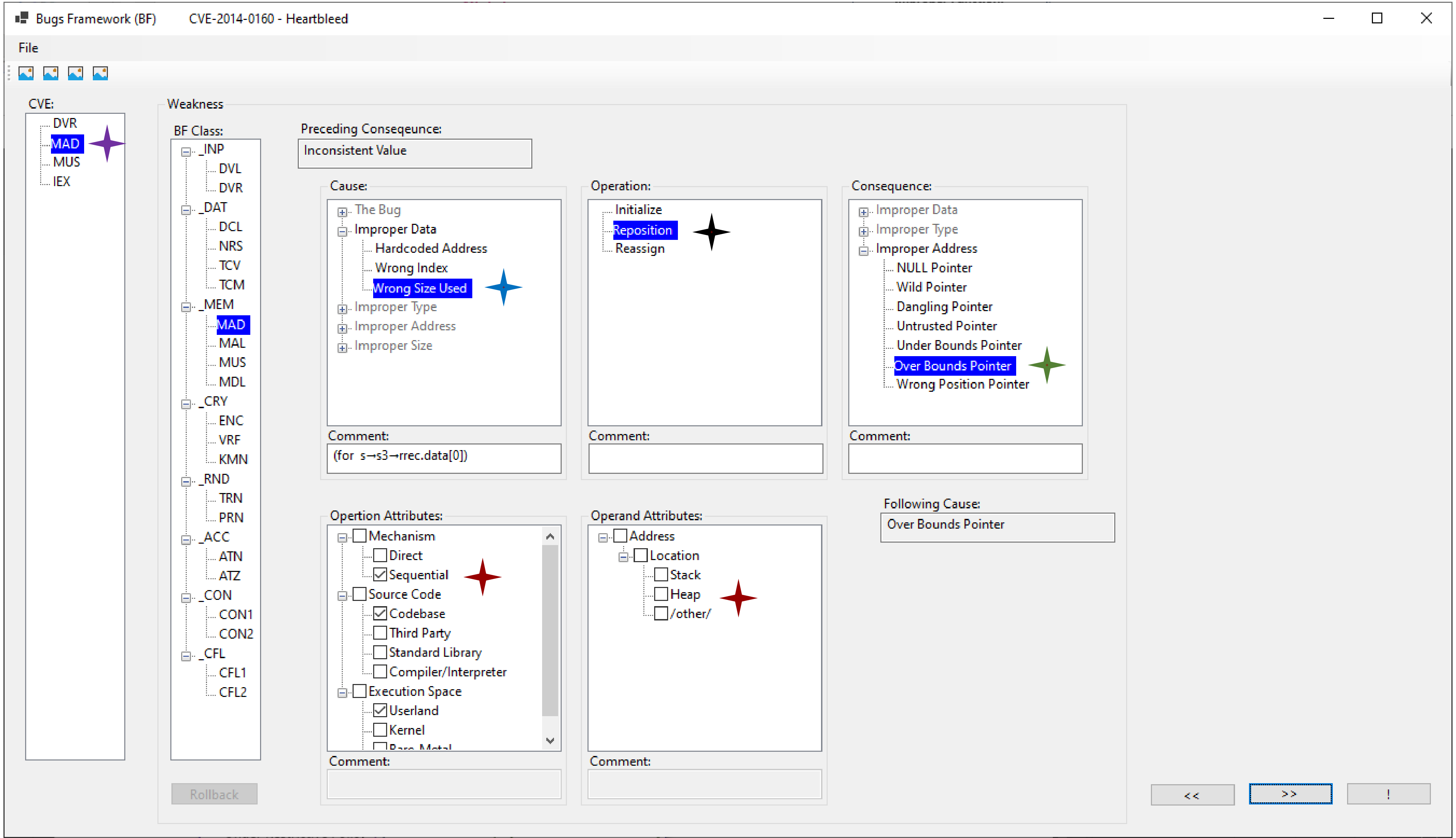
Consequence: Memory Error – Uninitialized Object; Not Cleared Object; NULL Pointer Dereference; Untrusted Pointer Dereferenc; Object Corruption; Type Confusion; Use After Free; **Buffer Overflow**; Buffer Underflow; Uninitialized Pointer Derefere

Following Cause: Buffer Overflow

Opertion Attributes:
- Mechanism: ☐ Direct ☑ Sequential
- Source Code: ☑ Codebase ☐ Third Party ☐ Standard Library ☐ Compiler/Interpreter
- Execution Space: ☑ Userland ☐ Kernel

Operand Attributes:
- Object Address – Span: ☐ Litle ☐ Moderate ☑ Huge
- Location: ☐ Stack ☑ Heap ☐ /other/

CVE-2014-0160 - Heartbleed.bfcve



CVE-2014-0160 - Heartbleed

CVE:
- DVR
- MAD
- MUS
- IEX

Weakness

BF Class:
- _INP
  - DVL
  - DVR
- _DAT
  - DCL
  - NRS
  - TCV
  - TCM
- _MEM
  - MAD
  - MAL
  - MUS
  - MDL
- _CRY
  - ENC
  - VRF
  - KMN
- _RND
  - TRN
  - PRN
- _ACC
  - ATN
  - ATZ
- _CON
  - CON1
  - CON2
- _CFL
  - CFL1
  - CFL2

Preceding Conseqeunce: Inconsistent Value

Cause:
- The Bug
- Improper Data
  - Hardcoded Address
  - Wrong Index
  - Wrong Size Used
- Improper Type
- Improper Address
- Improper Size

Comment: (for s→s3→rrec.data[0])

Operation:
- Initialize
- Reposition
- Reassign

Comment:

Consequence:
- Improper Data
- Improper Type
- Improper Address
  - NULL Pointer
  - Wild Pointer
  - Dangling Pointer
  - Untrusted Pointer
  - Under Bounds Pointer
  - Over Bounds Pointer
  - Wrong Position Pointer

Comment:

Opertion Attributes:
- □ Mechanism
  - □ Direct
  - ☑ Sequential
- □ Source Code
  - ☑ Codebase
  - □ Third Party
  - □ Standard Library
  - □ Compiler/Interpreter
- □ Execution Space
  - ☑ Userland
  - □ Kernel
  - □ Bare Metal

Comment:

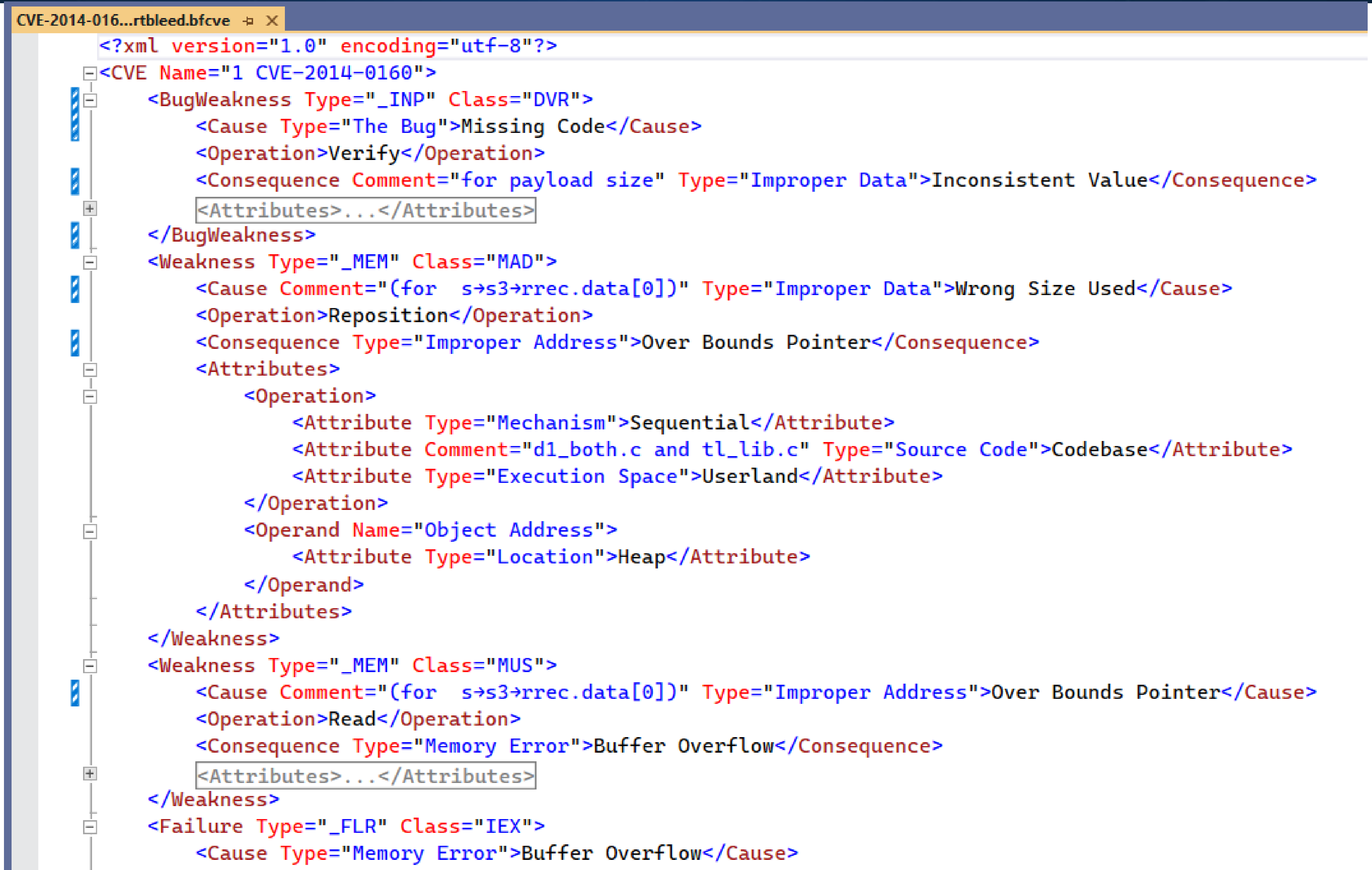
Operand Attributes:
- □ Address
  - □ Location
    - □ Stack
    - □ Heap
    - □ /other/

Comment:

Following Cause: Over Bounds Pointer

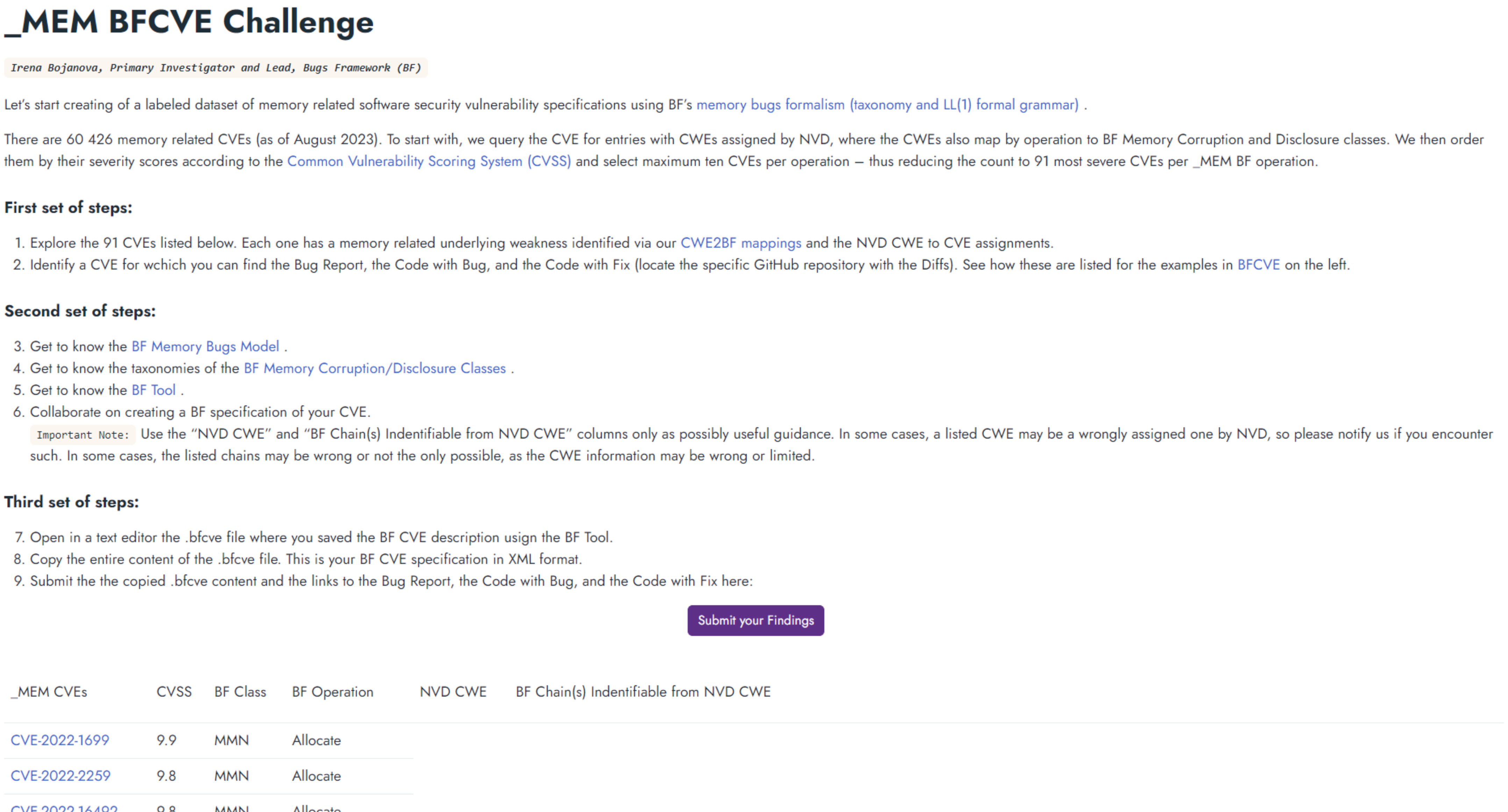
I. Bojanova, 2022

CVE-2014-0160 - Heartbleed.bfcve



```
<?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>
```

BF CVE Challenge



_MEM BFCVE Challenge

Irena Bojanova, Primary Investigator and Lead, Bugs Framework (BF)

Let's start creating of a labeled dataset of memory related software security vulnerability specifications using BF's memory bugs formalism (taxonomy and LL(1) formal grammar) .

There are 60 426 memory related CVEs (as of August 2023). To start with, we query the CVE for entries with CWEs assigned by NVD, where the CWEs also map by operation to BF Memory Corruption and Disclosure classes. We then order them by their severity scores according to the Common Vulnerability Scoring System (CVSS) and select maximum ten CVEs per operation – thus reducing the count to 91 most severe CVEs per _MEM BF operation.

First set of steps:

1. Explore the 91 CVEs listed below. Each one has a memory related underlying weakness identified via our CWE2BF mappings and the NVD CWE to CVE assignments.
2. Identify a CVE for wchich you can find the Bug Report, the Code with Bug, and the Code with Fix (locate the specific GitHub repository with the Diffs). See how these are listed for the examples in BFCVE on the left.

Second set of steps:

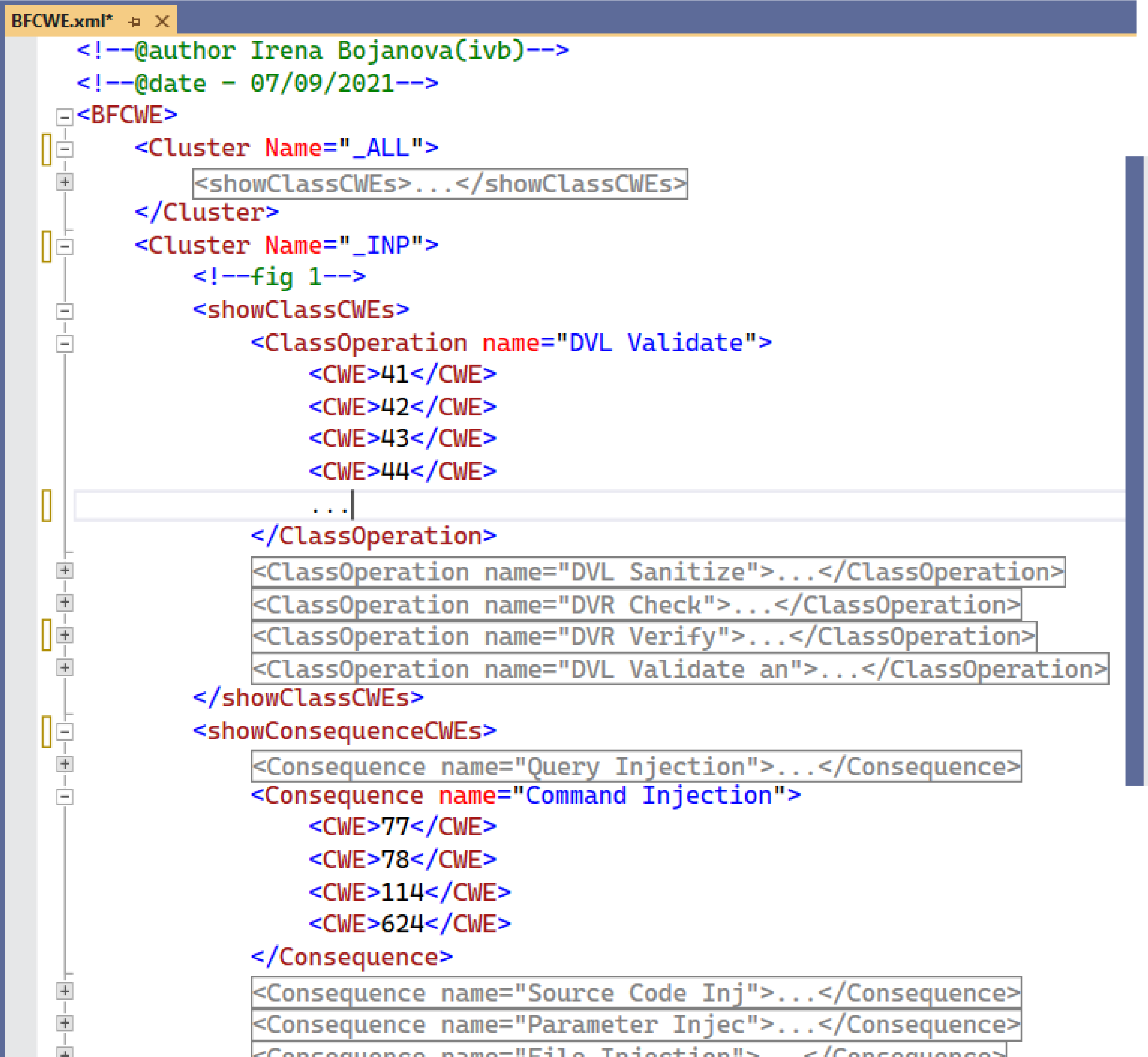
3. Get to know the BF Memory Bugs Model .
4. Get to know the taxonomies of the BF Memory Corruption/Disclosure Classes .
5. Get to know the BF Tool .
6. Collaborate on creating a BF specification of your CVE.
   `Important Note:` Use the "NVD CWE" and "BF Chain(s) Indentifiable from NVD CWE" columns only as possibly useful guidance. In some cases, a listed CWE may be a wrongly assigned one by NVD, so please notify us if you encounter such. In some cases, the listed chains may be wrong or not the only possible, as the CWE information may be wrong or limited.

Third set of steps:

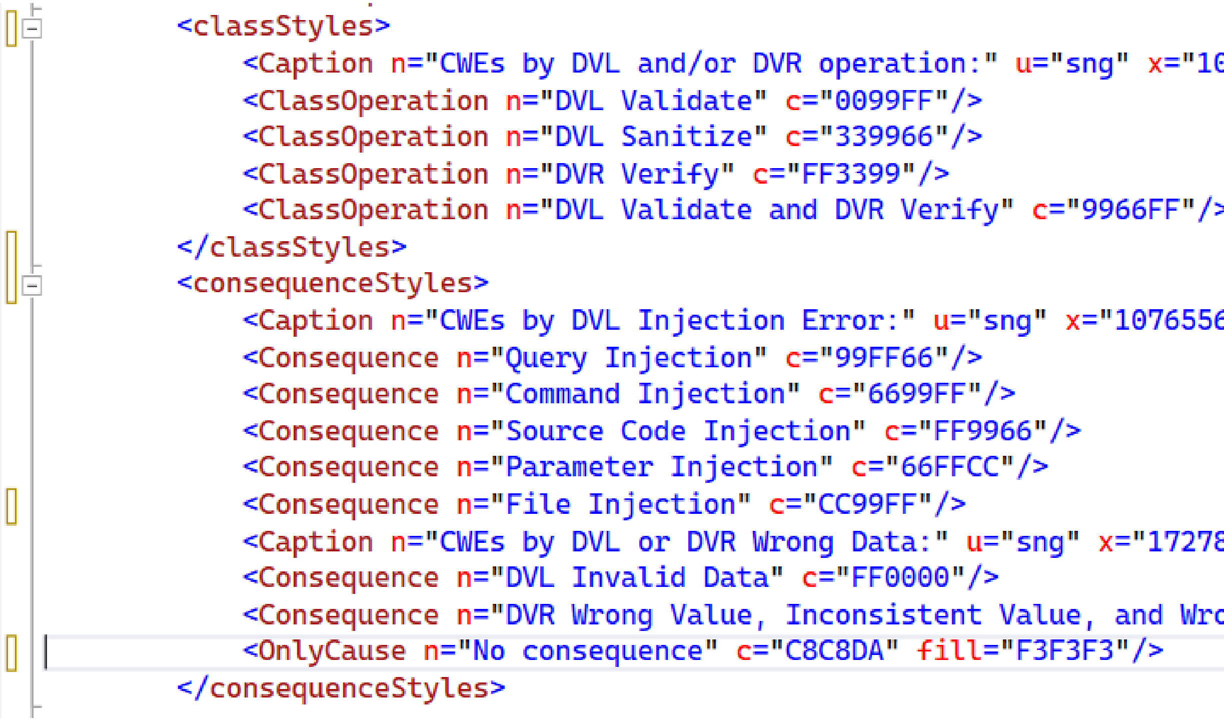
7. Open in a text editor the .bfcve file where you saved the BF CVE description usign the BF Tool.
8. Copy the entire content of the .bfcve file. This is your BF CVE specification in XML format.
9. Submit the the copied .bfcve content and the links to the Bug Report, the Code with Bug, and the Code with Fix here:

_MEM CVEs	CVSS	BF Class	BF Operation	NVD CWE	BF Chain(s) Indentifiable from NVD CWE
CVE-2022-1699	9.9	MMN	Allocate		
CVE-2022-2259	9.8	MMN	Allocate		
CVE-2022-16492	9.8	MMN	Allocate		

CWE mapped to BF – BFCWE.xml



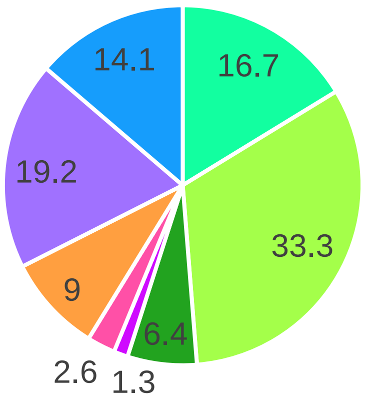
```
<!--@author Irena Bojanova(ivb)-->
<!--@date - 07/09/2021-->
<BFCWE>
    <Cluster Name="_ALL">
        <showClassCWEs>...</showClassCWEs>
    </Cluster>
    <Cluster Name="_INP">
        <!--fig 1-->
        <showClassCWEs>
            <ClassOperation name="DVL Validate">
                <CWE>41</CWE>
                <CWE>42</CWE>
                <CWE>43</CWE>
                <CWE>44</CWE>
                ...
            </ClassOperation>
            <ClassOperation name="DVL Sanitize">...</ClassOperation>
            <ClassOperation name="DVR Check">...</ClassOperation>
            <ClassOperation name="DVR Verify">...</ClassOperation>
            <ClassOperation name="DVL Validate an">...</ClassOperation>
        </showClassCWEs>
        <showConsequenceCWEs>
            <Consequence name="Query Injection">...</Consequence>
            <Consequence name="Command Injection">
                <CWE>77</CWE>
                <CWE>78</CWE>
                <CWE>114</CWE>
                <CWE>624</CWE>
            </Consequence>
            <Consequence name="Source Code Inj">...</Consequence>
            <Consequence name="Parameter Injec">...</Consequence>
            <Consequence name="File Injection">...</Consequence>
```



```
        <classStyles>
            <Caption n="CWEs by DVL and/or DVR operation:" u="sng" x="10
            <ClassOperation n="DVL Validate" c="0099FF"/>
            <ClassOperation n="DVL Sanitize" c="339966"/>
            <ClassOperation n="DVR Verify" c="FF3399"/>
            <ClassOperation n="DVL Validate and DVR Verify" c="9966FF"/>
        </classStyles>
        <consequenceStyles>
            <Caption n="CWEs by DVL Injection Error:" u="sng" x="1076556
            <Consequence n="Query Injection" c="99FF66"/>
            <Consequence n="Command Injection" c="6699FF"/>
            <Consequence n="Source Code Injection" c="FF9966"/>
            <Consequence n="Parameter Injection" c="66FFCC"/>
            <Consequence n="File Injection" c="CC99FF"/>
            <Caption n="CWEs by DVL or DVR Wrong Data:" u="sng" x="17278
            <Consequence n="DVL Invalid Data" c="FF0000"/>
            <Consequence n="DVR Wrong Value, Inconsistent Value, and Wro
            <OnlyCause n="No consequence" c="C8C8DA" fill="F3F3F3"/>
        </consequenceStyles>
```

Data Type CWEs by BF Operation

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

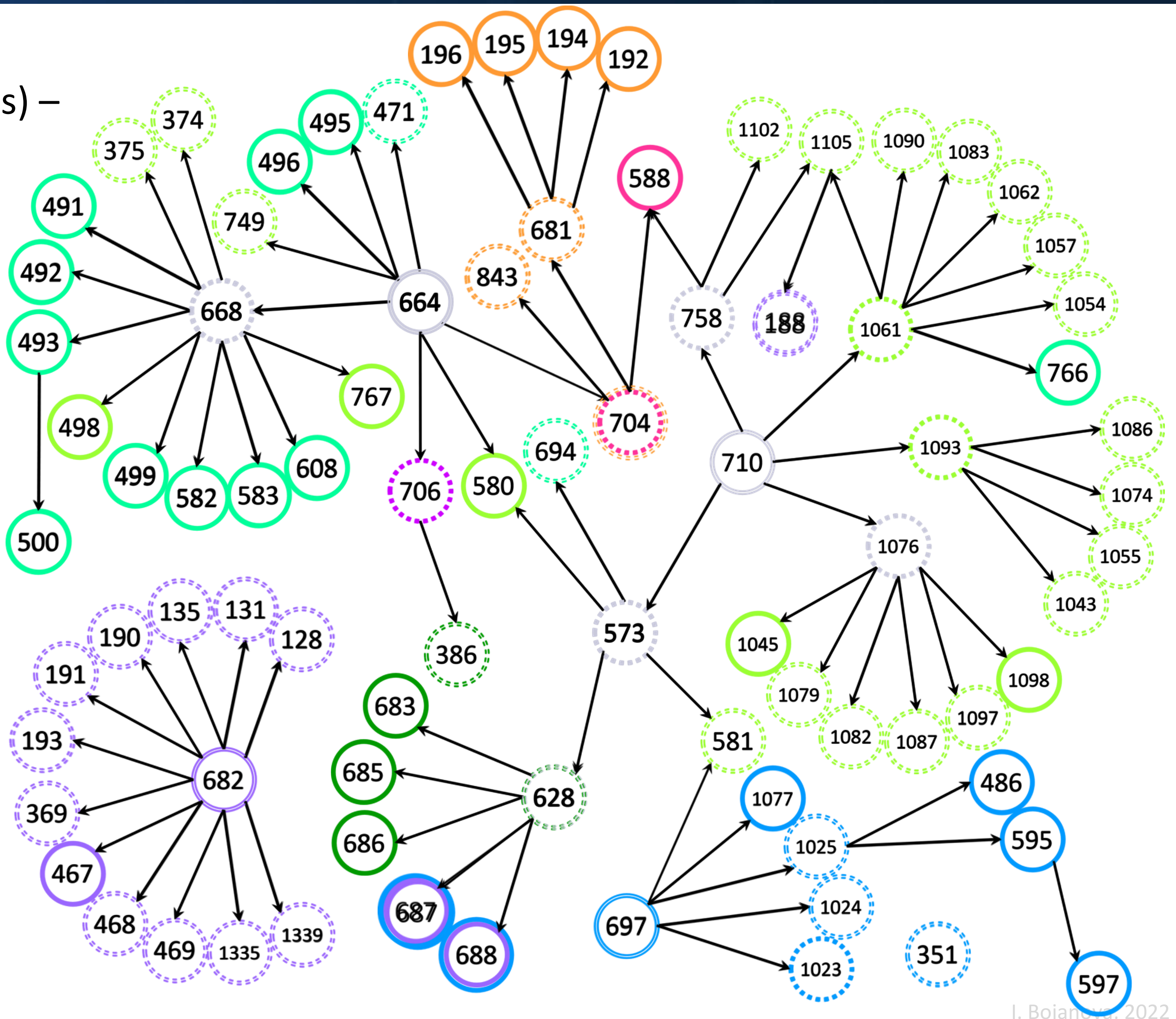


CWEs by DTC, NRS, TCV, and TCM operation:

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

CWEs by Abstraction:

○ Pillar
○ Class
○ Base
○ Variant



I. Bojanova, 2022

BF in ML & AI

Machine readable formats of:

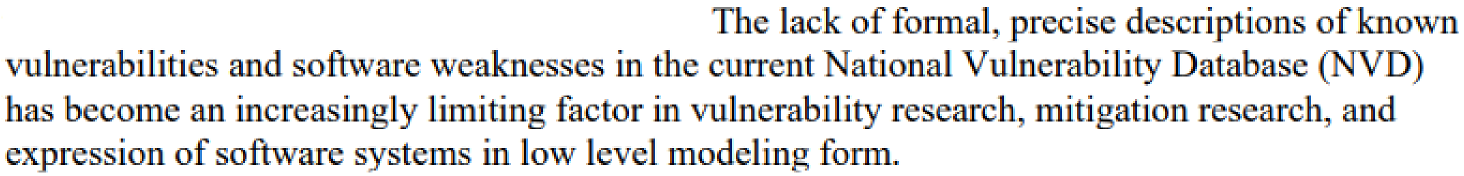
- BF taxonomy
- BF vulnerability descriptions
- CWEs to BF mappings

✓ Query and analyze sets of BF descriptions

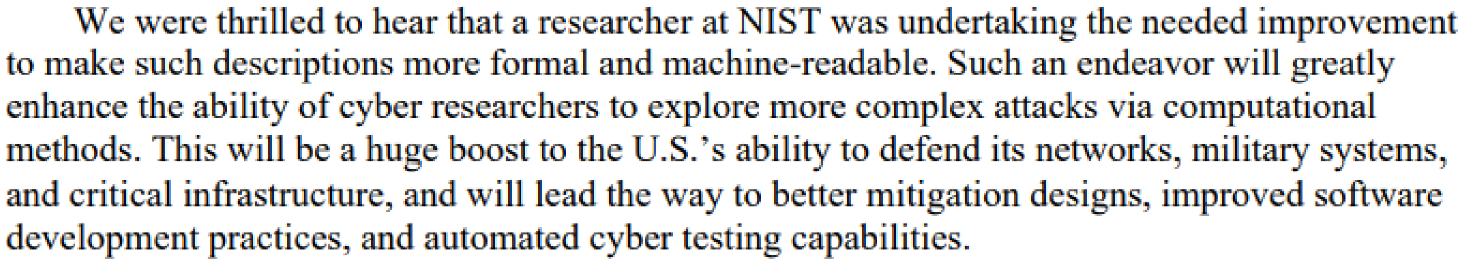
✓ NLP, ML, and AI projects related to software bugs/weaknesses, failures and risks.

BF in ML & AI

- JHU APL – Automated Vulnerability Testing via Executable Attack Graphs:
  - Chain vulnerabilities via logical directed graphs
  - Determine most mitigation “paths” with least changes
  - Detect user behavior prior to malicious effect

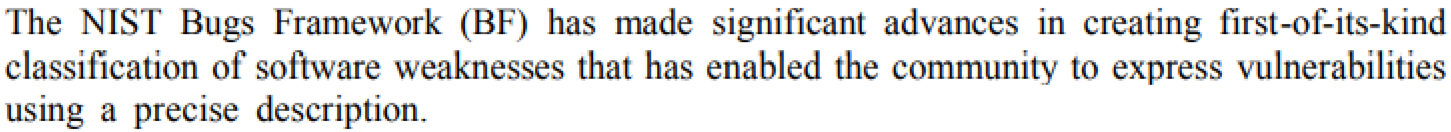


The lack of formal, precise descriptions of known vulnerabilities and software weaknesses in the current National Vulnerability Database (NVD) has become an increasingly limiting factor in vulnerability research, mitigation research, and expression of software systems in low level modeling form.

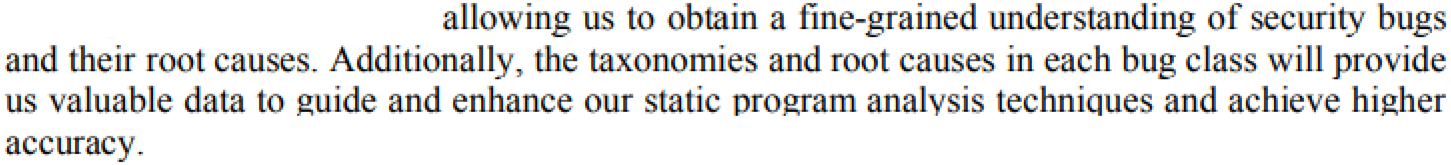


We were thrilled to hear that a researcher at NIST was undertaking the needed improvement to make such descriptions more formal and machine-readable. Such an endeavor will greatly enhance the ability of cyber researchers to explore more complex attacks via computational methods. This will be a huge boost to the U.S.’s ability to defend its networks, military systems, and critical infrastructure, and will lead the way to better mitigation designs, improved software development practices, and automated cyber testing capabilities.

- RIT Secure and Trustworthy Cyberspace (SaTC):



The NIST Bugs Framework (BF) has made significant advances in creating first-of-its-kind classification of software weaknesses that has enabled the community to express vulnerabilities using a precise description.

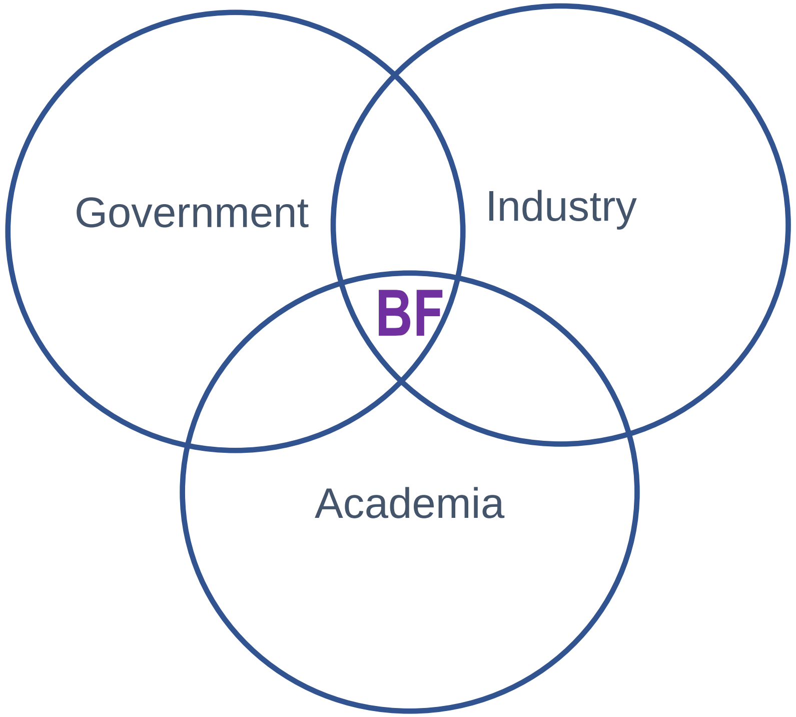


allowing us to obtain a fine-grained understanding of security bugs and their root causes. Additionally, the taxonomies and root causes in each bug class will provide us valuable data to guide and enhance our static program analysis techniques and achieve higher accuracy.

BF – Potential Impact

BF – Potential Impacts

- Allow precise communication about software bugs and weaknesses
- Help identify exploit mitigation techniques



Questions

Validation towards CWE

BF Class Related CWEs

- Identify CWEs:
  1. CWE Filtering
  2. Automated Extraction
  3. Manual Review

BF: https://samate.nist.gov/BF/
CWE: https://cwe.mitre.org/

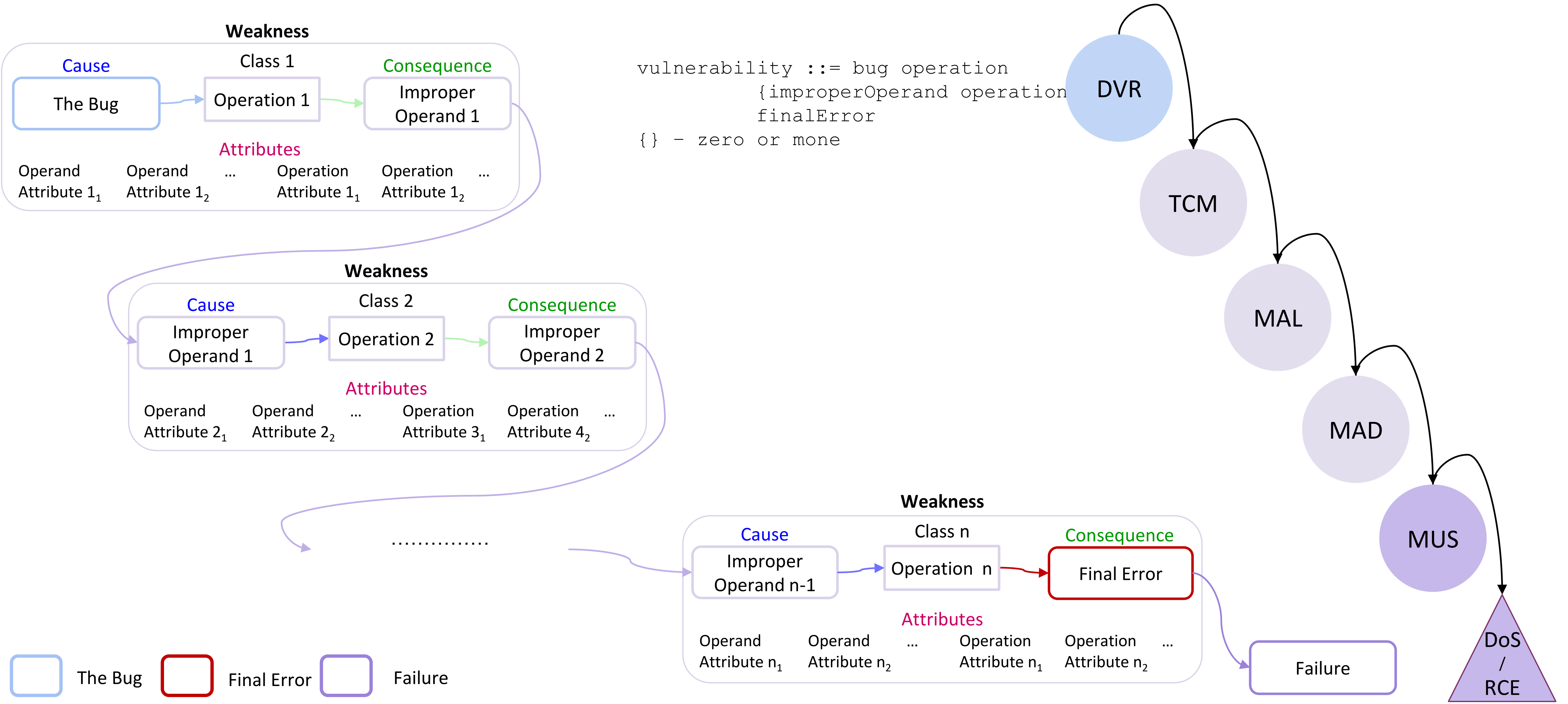
- BF Input/Output Bugs Classes – 161 CWEs:
  - ○ 80.7% – Input Validation Operation
  - ➢ 68.3% – Injection Error

- BF Data Type Bugs Classes – 78 CWEs:
  - ○ 50% Declaration/Definition Operation
  - ○ 33.3% Cast/Coerce Operation
  - ➢ 16% Access Error
  - ➢ 0.6% Type Compute Error

- BF Memory Bugs Classes 52 CWEs:
  - ○ 61.5% Initialize, Dereference, Read, Write, Clear Operations
  - ➢ 67.3% Memory Error

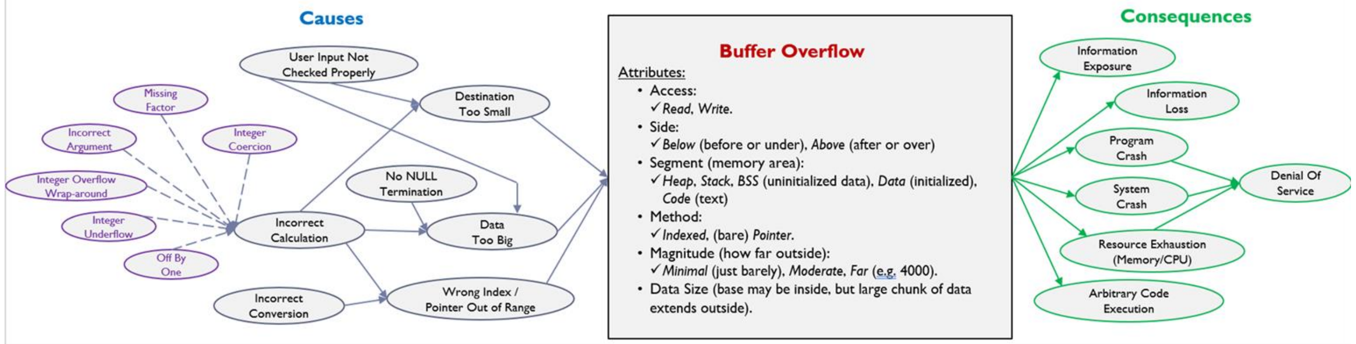
BF Model of Security Vulnerability

```
vulnerability ::= bug operation
         {improperOperand operation}
         finalError
{} - zero or mone
```



I. Bojanova, 2022

BF Early Work – Buffer Overflow



Buffer Overflow

Attributes:

- Access:
  ✓ *Read*, *Write*.
- Side:
  ✓ *Below* (before or under), *Above* (after or over)
- Segment (memory area):
  ✓ *Heap*, *Stack*, *BSS* (uninitialized data), *Data* (initialized), *Code* (text)
- Method:
  ✓ *Indexed*, (bare) *Pointer*.
- Magnitude (how far outside):
  ✓ *Minimal* (just barely), *Moderate*, *Far* (e.g. 4000).
- Data Size (base may be inside, but large chunk of data extends outside).

They Know Your Weaknesses – Do You?: Reintroducing Common Weakness Enumeration

Yan Wu, Bowling Green State University
Irena Bojanova, University of Maryland, Baltimore County
Yaacov Yesha, University of Maryland University College

Abstract: Knowing what makes your software systems vulnerable to attacks is critical, as software vulnerabilities hurt security, reliability, and availability of the system as a whole. The Common Weakness Enumeration (CWE), a community effort that provides the foundation for such knowledge, [illegible] rate and precise enough to serve as the common lan- [illegible] k and provide a common baseline for developers and

1.1 History of CWE

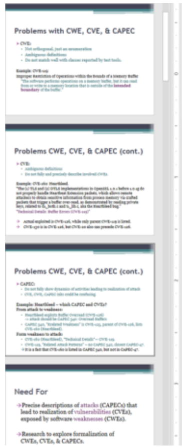
There have been several community efforts to leverage the existing large number of diverse real-world vulnerabilities. For example, an important step towards creating the needed collection of software weakness types was the establishment of the CVE (Common Vulnerabilities and Exposures) list [2] in 1999 by MITRE. Another important step from MITRE was creating the

Table 2. Buffer Overflow CWEs Attributes.

	before	after	either end	stack	heap
read	127	126	125		
write	124	120	123, 787	121	122
either r/w	786	788			

Where:

- access = either read/write
- outside = either before/below start or after/above



Problems with CWE, CVE, & CAPEC

- CWE:
  - Not orthogonal, just an enumeration
  - Ambiguous definitions
  - Do not match well with classes reported by test tools.

Example: 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."

Problems CWE, CVE, & CAPEC (cont.)

- CVE:
  - Ambiguous definitions
  - Do not fully and precisely describe involved CWEs.

Example: CVE-2014-0160: Heartbleed. [illegible]

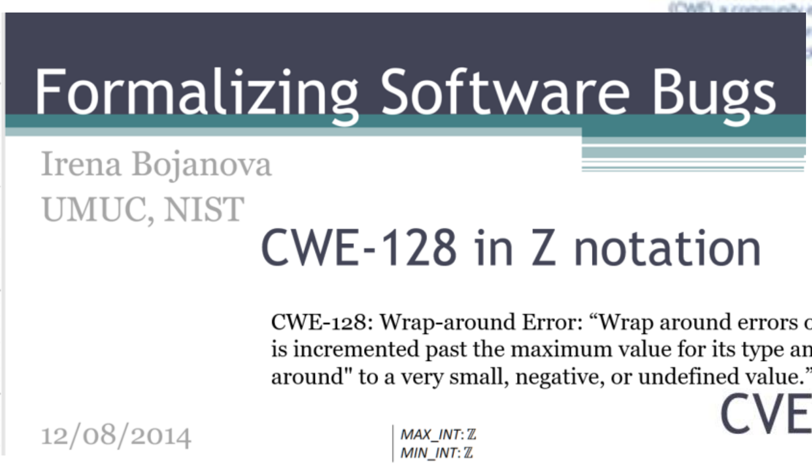
Problems CWE, CVE, & CAPEC (cont.)

- CAPEC:
  - Do not fully show dynamics of activities leading to realization of attack
  - CVE, CWE, CAPEC info could be confusing.

Example: Heartbleed – which CAPEC and CWEs? [illegible]

Need For

- Precise descriptions of attacks (CAPECs) that lead to realization of vulnerabilities (CVEs), exposed by software weaknesses (CWEs).
- Research to explore formalization of CWEs, CVEs, & CAPECs.



Formalizing Software Bugs

Irena Bojanova
UMUC, NIST

12/08/2014

CWE-128 in Z notation

CWE-128: Wrap-around Error: “Wrap around errors occur whenever a value is incremented past the maximum value for its type and therefore "wraps around" to a very small, negative, or undefined value.”

$$MAX\_INT : \mathbb{Z}$$
$$MIN\_INT : \mathbb{Z}$$

$$INT == \{ i : \mathbb{Z} \mid MIN\_INT \leq i \wedge i \leq MAX\_INT \}$$

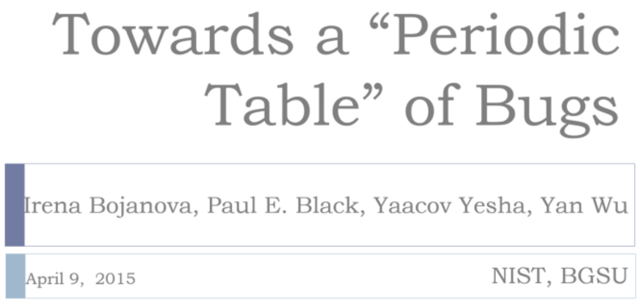
$$BAD\_INT : \mathbb{Z}$$
$$BAD\_INT < MIN\_INT \vee MAX\_INT < BAD\_INT$$

$$add, mul : INT \times INT \rightarrow INT \cup \{ BAD\_INT \}$$

CVE-2014-160/CAPEC-540 in CSP

```
channel network 2;
enum {payloadLength, payload, validPayload, invalidPayload};
Attacker() = network!payloadLength -> network!payload –
>network?payloadResponse->Attacker();
CWE_126() = network?payloadLength -> network?payload->
          (payloadLengthIsEqualTopayloadSize->network!validPayload->CWE_126()
          [] payloadLengthIsNotEqualTopayloadSize->network!invalidPayload ->
          CWE_126());

System() = Attacker() ||| CWE_126();
```

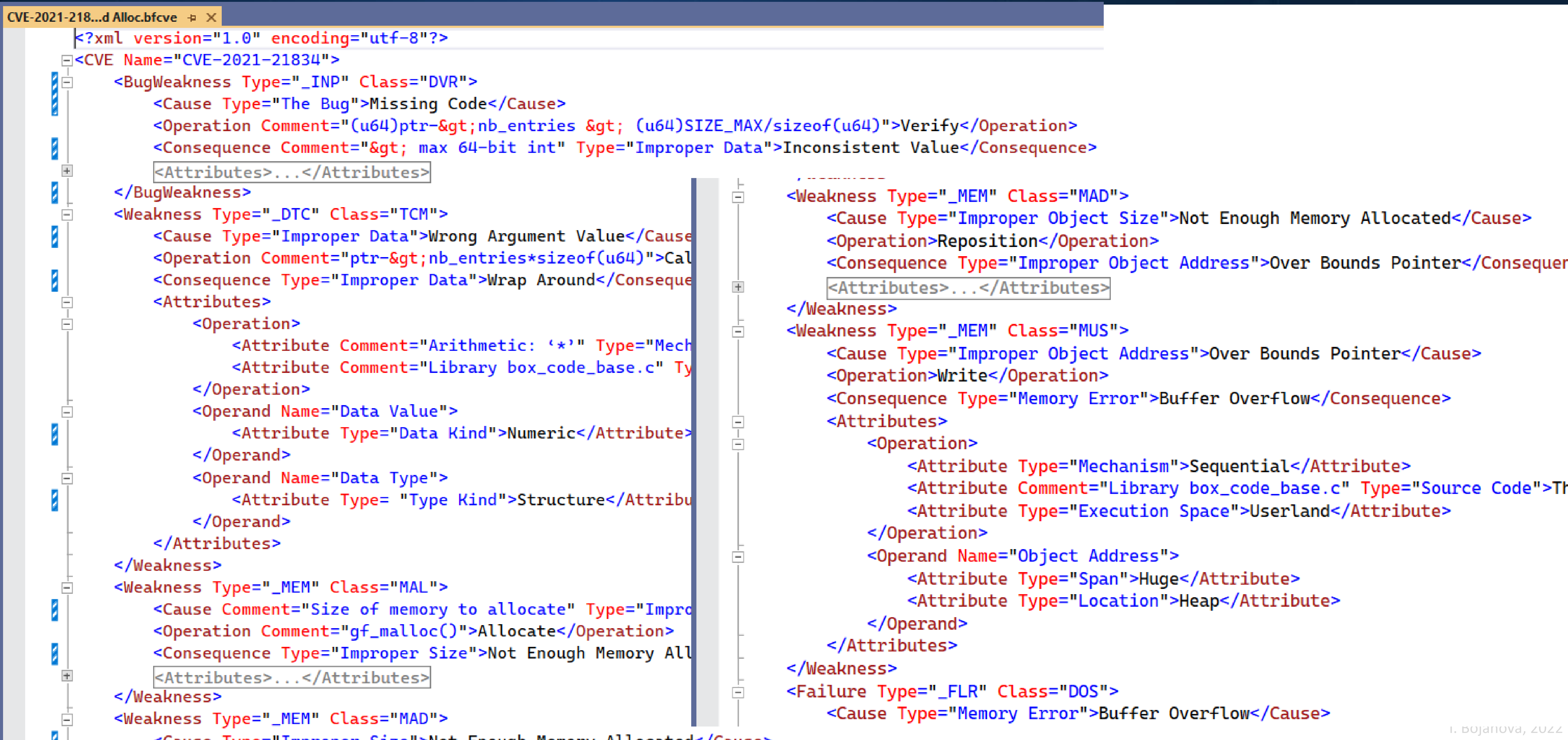


Towards a “Periodic Table” of Bugs

Irena Bojanova, Paul E. Black, Yaacov Yesha, Yan Wu

April 9, 2015 NIST, BGSU

CVE-2021-21834 - Bad Alloc.bfcve



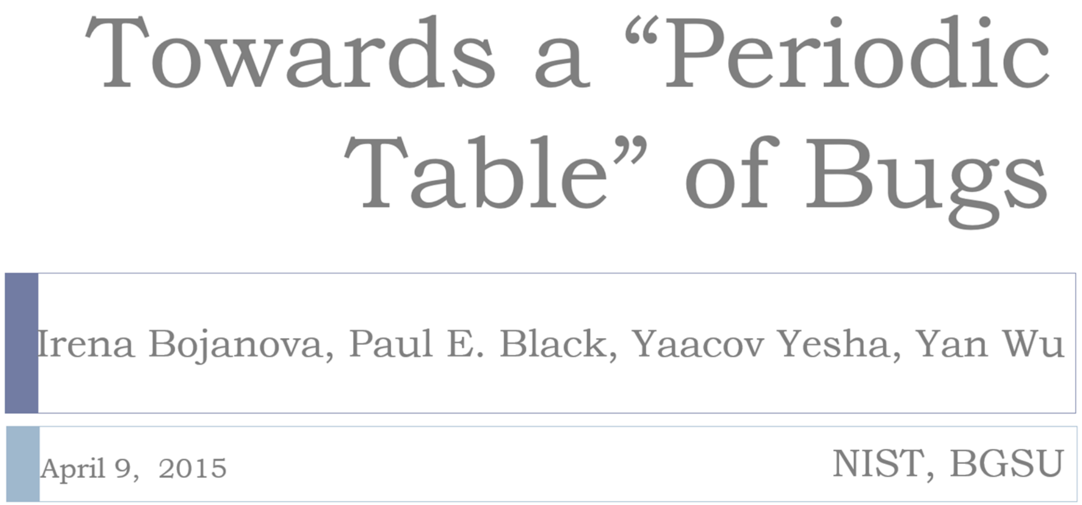
```
<?xml version="1.0" encoding="utf-8"?>
<CVE Name="CVE-2021-21834">
    <BugWeakness Type="_INP" Class="DVR">
        <Cause Type="The Bug">Missing Code</Cause>
        <Operation Comment="(u64)ptr-&gt;nb_entries &gt; (u64)SIZE_MAX/sizeof(u64)">Verify</Operation>
        <Consequence Comment="&gt; max 64-bit int" Type="Improper Data">Inconsistent Value</Consequence>
        <Attributes>...</Attributes>
    </BugWeakness>
    <Weakness Type="_DTC" Class="TCM">
        <Cause Type="Improper Data">Wrong Argument Value</Cause
        <Operation Comment="ptr-&gt;nb_entries*sizeof(u64)">Cal
        <Consequence Type="Improper Data">Wrap Around</Conseque
        <Attributes>
            <Operation>
                <Attribute Comment="Arithmetic: '*'" Type="Mech
                <Attribute Comment="Library box_code_base.c" Ty
            </Operation>
            <Operand Name="Data Value">
                <Attribute Type="Data Kind">Numeric</Attribute>
            </Operand>
            <Operand Name="Data Type">
                <Attribute Type= "Type Kind">Structure</Attribu
            </Operand>
        </Attributes>
    </Weakness>
    <Weakness Type="_MEM" Class="MAL">
        <Cause Comment="Size of memory to allocate" Type="Impro
        <Operation Comment="gf_malloc()">Allocate</Operation>
        <Consequence Type="Improper Size">Not Enough Memory All
        <Attributes>...</Attributes>
    </Weakness>
    <Weakness Type="_MEM" Class="MAD">
        <Cause Type="Improper Size">Not Enough Memory Allocated</Cause>
```

```
    </Weakness>
    <Weakness Type="_MEM" Class="MAD">
        <Cause Type="Improper Object Size">Not Enough Memory Allocated</Cause>
        <Operation>Reposition</Operation>
        <Consequence Type="Improper Object Address">Over Bounds Pointer</Consequen
        <Attributes>...</Attributes>
    </Weakness>
    <Weakness Type="_MEM" Class="MUS">
        <Cause Type="Improper Object Address">Over Bounds Pointer</Cause>
        <Operation>Write</Operation>
        <Consequence Type="Memory Error">Buffer Overflow</Consequence>
        <Attributes>
            <Operation>
                <Attribute Type="Mechanism">Sequential</Attribute>
                <Attribute Comment="Library box_code_base.c" Type="Source Code">Th
                <Attribute Type="Execution Space">Userland</Attribute>
            </Operation>
            <Operand Name="Object Address">
                <Attribute Type="Span">Huge</Attribute>
                <Attribute Type="Location">Heap</Attribute>
            </Operand>
        </Attributes>
    </Weakness>
    <Failure Type="_FLR" Class="DOS">
        <Cause Type="Memory Error">Buffer Overflow</Cause>
```

I. Bojanova, 2022

BF Early Work – Heartbleed

- Heartbleed buffer overflow is:
  - caused by *Data Too Big*
  - because of *User Input not Checked Properly*
  - where there was a *Read that was After the End that was Far Outside*
  - of a buffer in the *Heap*
  - which may be exploited for *Information Exposure*

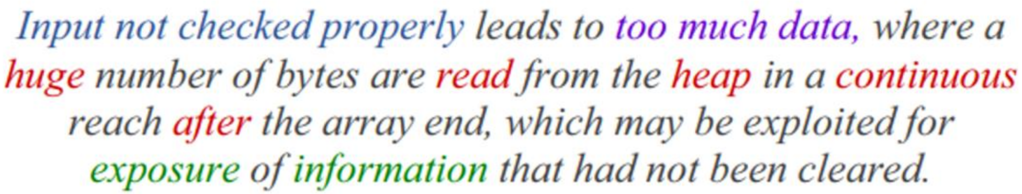


Towards a "Periodic Table" of Bugs

Irena Bojanova, Paul E. Black, Yaacov Yesha, Yan Wu

April 9, 2015

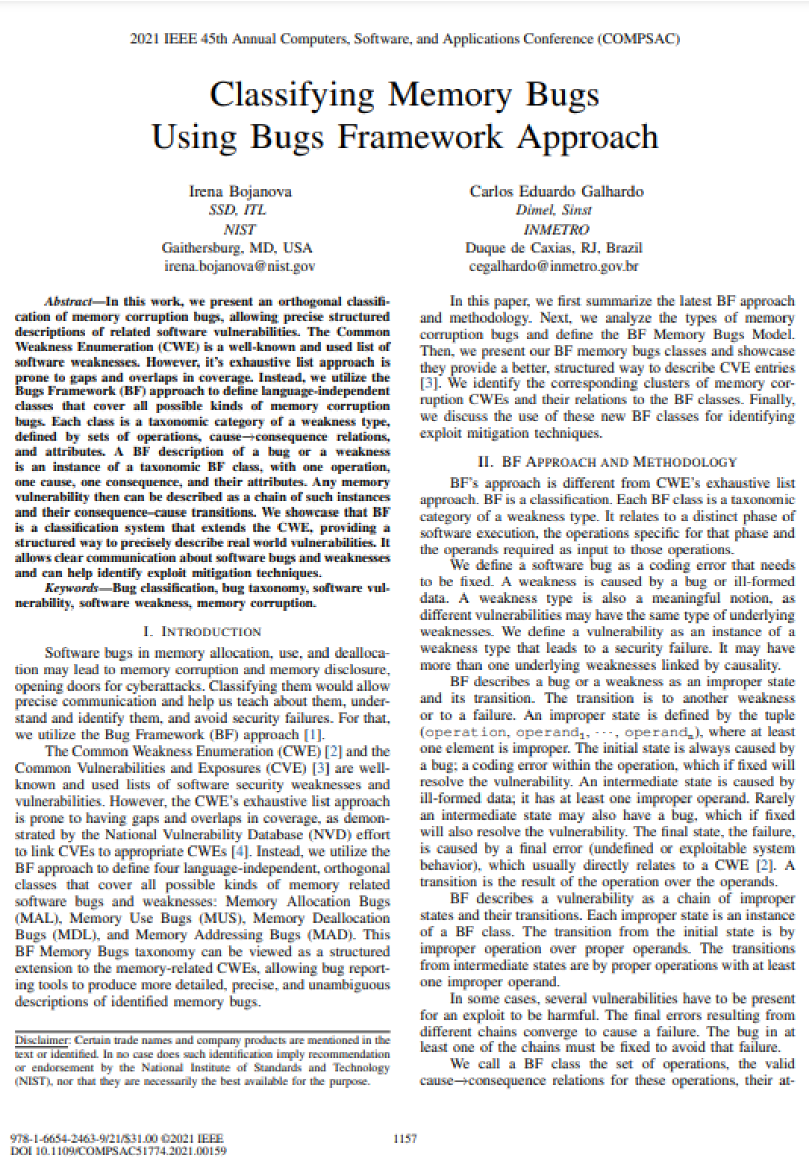
NIST, BGSU



Input not checked properly leads to 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.

Bojanova, I., Black, P., Yesha, Y. and Wu, Y. (2016), The Bugs Framework (BF): A Structured Approach to Express Bugs, IEEE International Conference on Software Quality, Reliability & Security (QRS 2016), Viena, AT,

Latest BF Publications



2021 IEEE 45th Annual Computers, Software, and Applications Conference (COMPSAC)

Classifying Memory Bugs Using Bugs Framework Approach

Irena Bojanova
SSD, ITL
NIST
Gaithersburg, MD, USA
irena.bojanova@nist.gov

Carlos Eduardo Galhardo
Dimel, Sinst
INMETRO
Duque de Caxias, RJ, Brazil
cegalhardo@inmetro.gov.br

***Abstract*—In this work, we present an orthogonal classification of memory corruption bugs, allowing precise structured descriptions of related software vulnerabilities. The Common Weakness Enumeration (CWE) is a well-known and used list of software weaknesses. However, it's exhaustive list approach is prone to gaps and overlaps in coverage. Instead, we utilize the Bugs Framework (BF) approach to define language-independent classes that cover all possible kinds of memory corruption bugs. Each class is a taxonomic category of a weakness type, defined by sets of operations, cause→consequence relations, and attributes. A BF description of a bug or a weakness is an instance of a taxonomic BF class, with one operation, one cause, one consequence, and their attributes. Any memory vulnerability then can be described as a chain of such instances and their consequence–cause transitions. We showcase that BF is a classification system that extends the CWE, providing a structured way to precisely describe real world vulnerabilities. It allows clear communication about software bugs and weaknesses and can help identify exploit mitigation techniques.**

***Keywords*—Bug classification, bug taxonomy, software vulnerability, software weakness, memory corruption.**

I. Introduction

Software bugs in memory allocation, use, and deallocation may lead to memory corruption and memory disclosure, opening doors for cyberattacks. Classifying them would allow precise communication and help us teach about them, understand and identify them, and avoid security failures. For that, we utilize the Bug Framework (BF) approach [1].

The Common Weakness Enumeration (CWE) [2] and the Common Vulnerabilities and Exposures (CVE) [3] are well-known and used lists of software security weaknesses and vulnerabilities. However, the CWE's exhaustive list approach is prone to having gaps and overlaps in coverage, as demonstrated by the National Vulnerability Database (NVD) effort to link CVEs to appropriate CWEs [4]. Instead, we utilize the BF approach to define four language-independent, orthogonal classes that cover all possible kinds of memory related software bugs and weaknesses: Memory Allocation Bugs (MAL), Memory Use Bugs (MUS), Memory Deallocation Bugs (MDL), and Memory Addressing Bugs (MAD). This BF Memory Bugs taxonomy can be viewed as a structured extension to the memory-related CWEs, allowing bug reporting tools to produce more detailed, precise, and unambiguous descriptions of identified memory bugs.

In this paper, we first summarize the latest BF approach and methodology. Next, we analyze the types of memory corruption bugs and define the BF Memory Bugs Model. Then, we present our BF memory bugs classes and showcase they provide a better, structured way to describe CVE entries [3]. We identify the corresponding clusters of memory corruption CWEs and their relations to the BF classes. Finally, we discuss the use of these new BF classes for identifying exploit mitigation techniques.

II. BF Approach and Methodology

BF's approach is different from CWE's exhaustive list approach. BF is a classification. Each BF class is a taxonomic category of a weakness type. It relates to a distinct phase of software execution, the operations specific for that phase and the operands required as input to those operations.

We define a software bug as a coding error that needs to be fixed. A weakness is caused by a bug or ill-formed data. A weakness type is also a meaningful notion, as different vulnerabilities may have the same type of underlying weaknesses. We define a vulnerability as an instance of a weakness type that leads to a security failure. It may have more than one underlying weaknesses linked by causality.

BF describes a bug or a weakness as an improper state and its transition. The transition is to another weakness or to a failure. An improper state is defined by the tuple $(\texttt{operation}, \texttt{operand}_1, \cdots, \texttt{operand}_n)$, where at least one element is improper. The initial state is always caused by a bug; a coding error within the operation, which if fixed will resolve the vulnerability. An intermediate state is caused by ill-formed data; it has at least one improper operand. Rarely an intermediate state may also have a bug, which if fixed will also resolve the vulnerability. The final state, the failure, is caused by a final error (undefined or exploitable system behavior), which usually directly relates to a CWE [2]. A transition is the result of the operation over the operands.

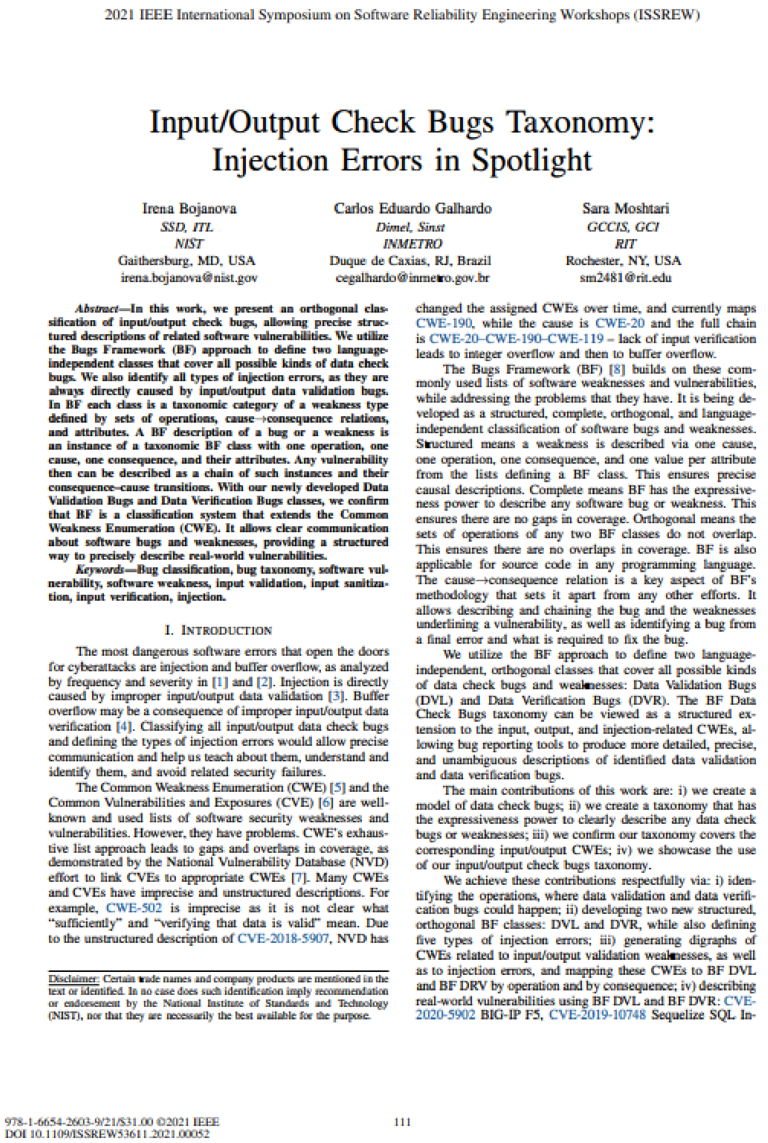
BF describes a vulnerability as a chain of improper states and their transitions. Each improper state is an instance of a BF class. The transition from the initial state is by improper operation over proper operands. The transitions from intermediate states are by proper operations with at least one improper operand.

In some cases, several vulnerabilities have to be present for an exploit to be harmful. The final errors resulting from different chains converge to cause a failure. The bug in at least one of the chains must be fixed to avoid that failure.

We call a BF class the set of operations, the valid cause→consequence relations for these operations, their at-

Disclaimer: Certain trade names and company products are mentioned in the text or identified. In no case does such identification imply recommendation or endorsement by the National Institute of Standards and Technology (NIST), nor that they are necessarily the best available for the purpose.

978-1-6654-2463-9/21/$31.00 ©2021 IEEE
DOI 10.1109/COMPSAC51774.2021.00159



2021 IEEE International Symposium on Software Reliability Engineering Workshops (ISSREW)

Input/Output Check Bugs Taxonomy: Injection Errors in Spotlight

Irena Bojanova
SSD, ITL
NIST
Gaithersburg, MD, USA
irena.bojanova@nist.gov

Carlos Eduardo Galhardo
Dimel, Sinst
INMETRO
Duque de Caxias, RJ, Brazil
cegalhardo@inmetro.gov.br

Sara Moshtari
GCCIS, GCI
RIT
Rochester, NY, USA
sm2481@rit.edu

***Abstract*—In this work, we present an orthogonal classification of input/output check bugs, allowing precise structured descriptions of related software vulnerabilities. We utilize the Bugs Framework (BF) approach to define two language-independent classes that cover all possible kinds of data check bugs. We also identify all types of injection errors, as they are always directly caused by input/output data validation bugs. In BF each class is a taxonomic category of a weakness type defined by sets of operations, cause→consequence relations, and attributes. A BF description of a bug or a weakness is an instance of a taxonomic BF class with one operation, one cause, one consequence, and their attributes. Any vulnerability then can be described as a chain of such instances and their consequence–cause transitions. With our newly developed Data Validation Bugs and Data Verification Bugs classes, we confirm that BF is a classification system that extends the Common Weakness Enumeration (CWE). It allows clear communication about software bugs and weaknesses, providing a structured way to precisely describe real-world vulnerabilities.**

***Keywords*—Bug classification, bug taxonomy, software vulnerability, software weakness, input validation, input sanitization, input verification, injection.**

I. Introduction

The most dangerous software errors that open the doors for cyberattacks are injection and buffer overflow, as analyzed by frequency and severity in [1] and [2]. Injection is directly caused by improper input/output data validation [3]. Buffer overflow may be a consequence of improper input/output data verification [4]. Classifying all input/output data check bugs and defining the types of injection errors would allow precise communication and help us teach about them, understand and identify them, and avoid related security failures.

The Common Weakness Enumeration (CWE) [5] and the Common Vulnerabilities and Exposures (CVE) [6] are well-known and used lists of software security weaknesses and vulnerabilities. However, they have problems. CWE's exhaustive list approach leads to gaps and overlaps in coverage, as demonstrated by the National Vulnerability Database (NVD) effort to link CVEs to appropriate CWEs [7]. Many CWEs and CVEs have imprecise and unstructured descriptions. For example, CWE-502 is imprecise as it is not clear what "sufficiently" and "verifying that data is valid" mean. Due to the unstructured description of CVE-2018-5907, NVD has changed the assigned CWEs over time, and currently maps CWE-190, while the cause is CWE-20 and the full chain is CWE-20–CWE-190–CWE-119 – lack of input verification leads to integer overflow and then to buffer overflow.

The Bugs Framework (BF) [8] builds on these commonly used lists of software weaknesses and vulnerabilities, while addressing the problems that they have. It is being developed as a structured, complete, orthogonal, and language-independent classification of software bugs and weaknesses. Structured means a weakness is described via one cause, one operation, one consequence, and one value per attribute from the lists defining a BF class. This ensures precise causal descriptions. Complete means BF has the expressiveness power to describe any software bug or weakness. This ensures there are no gaps in coverage. Orthogonal means the sets of operations of any two BF classes do not overlap. This ensures there are no overlaps in coverage. BF is also applicable for source code in any programming language. The cause→consequence relation is a key aspect of BF's methodology that sets it apart from any other efforts. It allows describing and chaining the bug and the weaknesses underlining a vulnerability, as well as identifying a bug from a final error and what is required to fix the bug.

We utilize the BF approach to define two language-independent, orthogonal classes that cover all possible kinds of data check bugs and weaknesses: Data Validation Bugs (DVL) and Data Verification Bugs (DVR). The BF Data Check Bugs taxonomy can be viewed as a structured extension to the input, output, and injection-related CWEs, allowing bug reporting tools to produce more detailed, precise, and unambiguous descriptions of identified data validation and data verification bugs.

The main contributions of this work are: i) we create a model of data check bugs; ii) we create a taxonomy that has the expressiveness power to clearly describe any data check bugs or weaknesses; iii) we confirm our taxonomy covers the corresponding input/output CWEs; iv) we showcase the use of our input/output check bugs taxonomy.

We achieve these contributions respectfully via: i) identifying the operations, where data validation and data verification bugs could happen; ii) developing two new structured, orthogonal BF classes: DVL and DVR, while also defining five types of injection errors; iii) generating digraphs of CWEs related to input/output validation weaknesses, as well as to injection errors, and mapping these CWEs to BF DVL and BF DRV by operation and by consequence; iv) describing real-world vulnerabilities using BF DVL and BF DVR: CVE-2020-5902 BIG-IP F5, CVE-2019-10748 Sequelize SQL In-

Disclaimer: Certain trade names and company products are mentioned in the text or identified. In no case does such identification imply recommendation or endorsement by the National Institute of Standards and Technology (NIST), nor that they are necessarily the best available for the purpose.

978-1-6654-2603-9/21/$31.00 ©2021 IEEE
DOI 10.1109/ISSREW53611.2021.00052

BF Contact

Irena Bojanova: irena.bojanova@nist.gov

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