NVD–BF (or NVD^{BF})

Alt.

Formal Vulnerability Classifications Platform to Accelerate AI and FM Cybersecurity R&D

Concept Pitch
IMS FY 2027
January 22, 2026

NVD – National Vulnerability Database
BF – NIST Bugs Framework
AI – Artificial Intelligence
FM – Formal Methods
R&D – Research and Development

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

Irena Bojanova
(ITL/775)

Reality → Vulnerability R&D → Cybersecurity

NIST

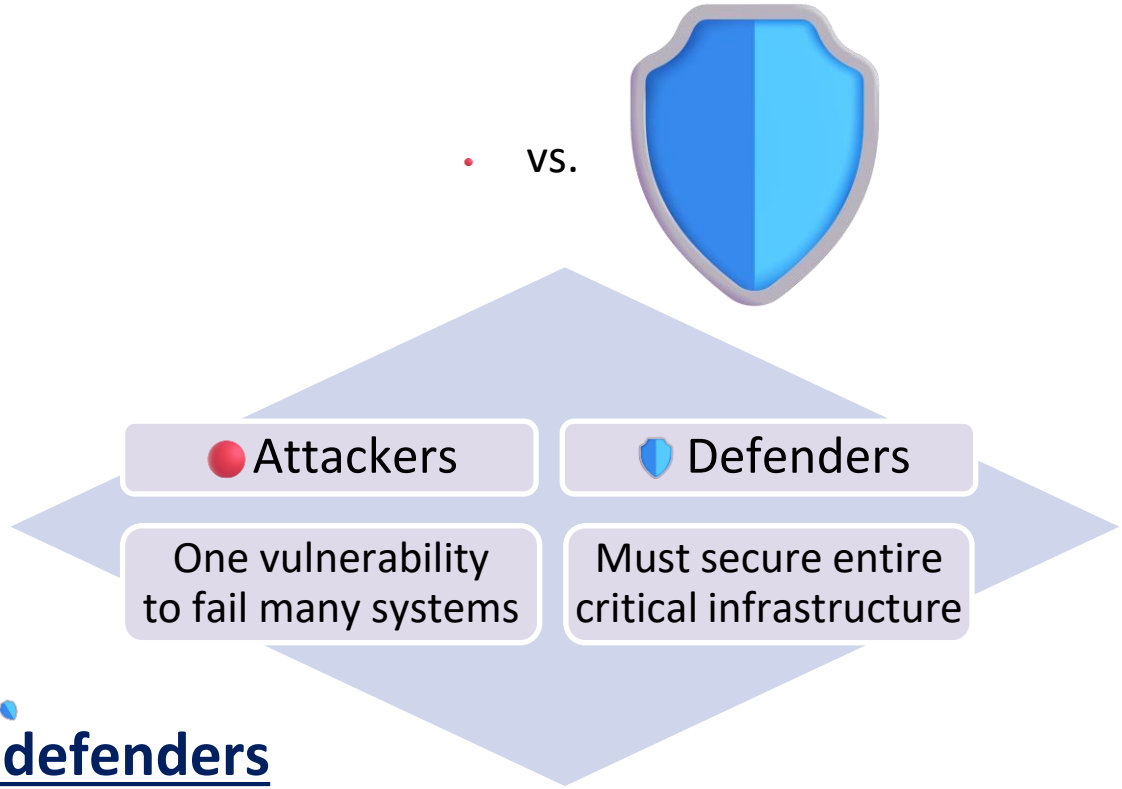
- Vulnerability exploits – primary infection vector

Mandiant, 2025

- Finding/fixing **code** defects: **$600+ B**

CISQ, 2022

- Defender's Paradox



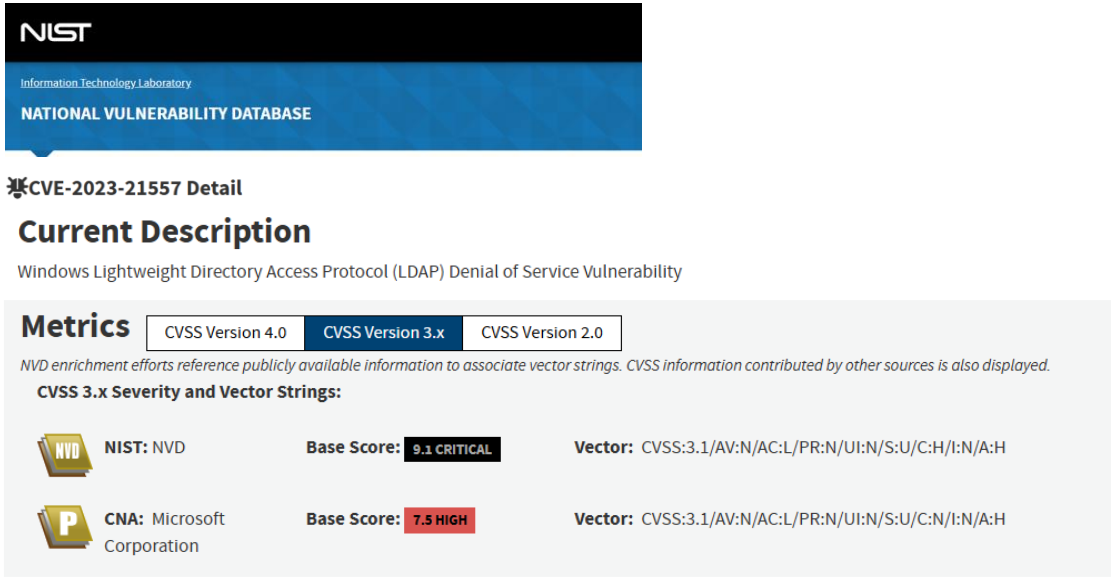
Accelerate vulnerability R&D to empower defenders

- ✓ AI-based agents and systems
- ✓ Formal verification methods

NIST Priority: Secure US critical infrastructure.

Advance Vulnerability Analytics

- Vulnerability repositories
  - NVD, KEV
  - CVE (& CWE)
- Problem – limited analytics
  - Narrative descriptions
  - Static, subjective data
  - Lack of labeled datasets



Multi-dimensional vulnerability formalism?

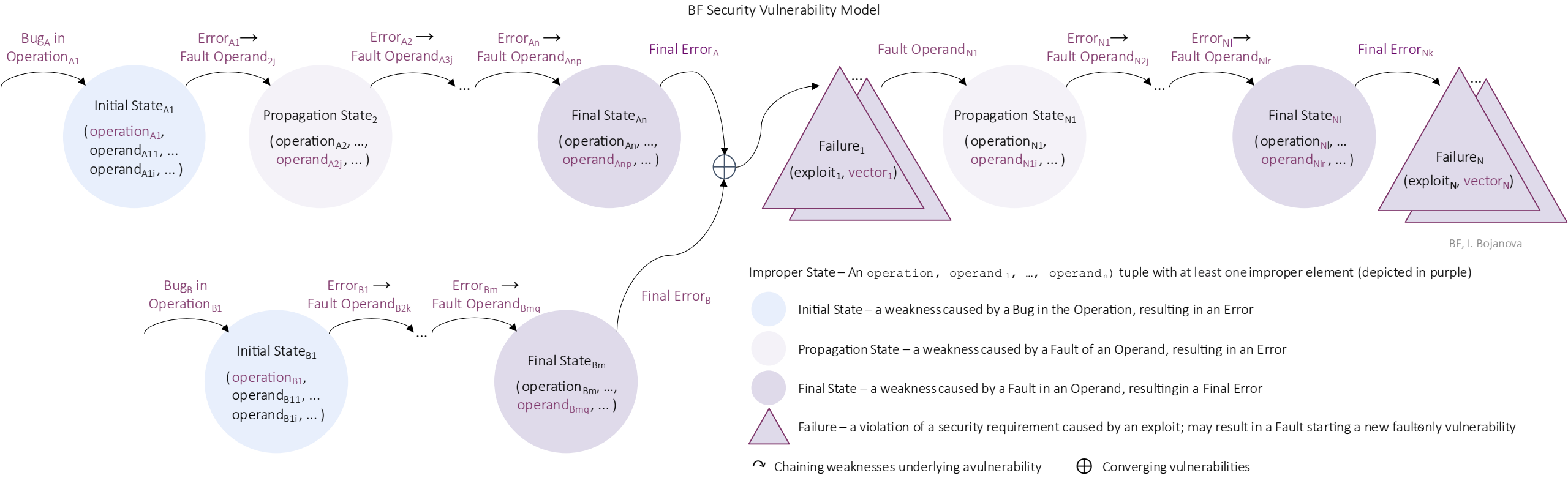
- ✓ Formal specification
- ✓ Secure coding principles
- ✓ Automatic generation
- ✓ Dynamic vulnerability classifications
- ✓ Contextual connections
- ✓ Analytics and scoring

NIST Priority: AI-based agents to secure US critical infrastructure.

Utilize

Bugs Framework (BF)

- Multi-dimension classification of security bugs and related to them faults
- Understand vulnerabilities – weakness causation, propagation, convergence



Pending Patent: U.S. Patent Application No. PCT/US2025/038662, Bugs Framework (BF) – A System for Formal Specification of Cybersecurity Weaknesses and Vulnerabilities, Definition of Secure Coding Principles, and Generation of Weakness and Vulnerability Datasets and Vulnerability Classifications. Owner: The Government of the Unites States, as represented by the Secretary of Commerce. Inventor: Irena Bojanova

NVD–BF: Heartbleed

Example

NATIONAL VULNERABILITY DATABASE

CVE-2014-0160 Detail

Description

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.

Metrics

CVSS Version 4.0 | CVSS Version 3.x | CVSS Version 2.0

NVD enrichment efforts reference publicly available information to associate vector strings. CVSS information contributed by other sources is also displayed.

CVSS 3.x Severity and Vector Strings:

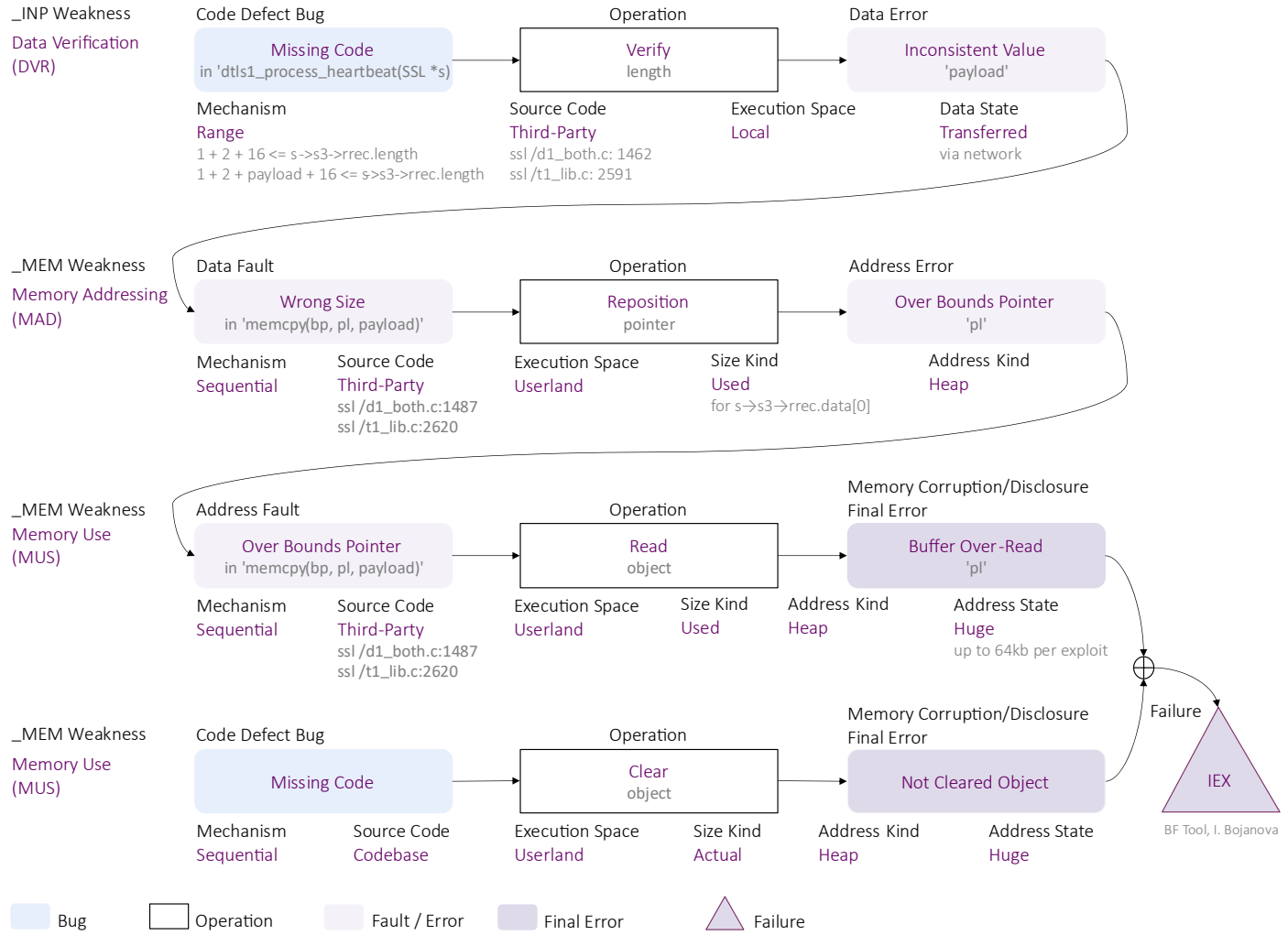
NIST: NVD **Base Score:** 7.5 HIGH **Vector:** CVSS:3.1/AV:N/AC:L/PR:N/UI:N/S:U/C:H/I:N/A:N

ADP: CISA-ADP **Base Score:** 7.5 HIGH **Vector:** CVSS:3.1/AV:N/AC:L/PR:N/UI:N/S:U/C:H/I:N/A:N

Weakness Enumeration

CWE-ID	CWE Name	Source
CWE-125	Out-of-bounds Read	NIST CISA-ADP

BF Specification of CVE -2014-0160 – Heartbleed in OpenSSL v1.0.1 before v1.0.1g



Augment

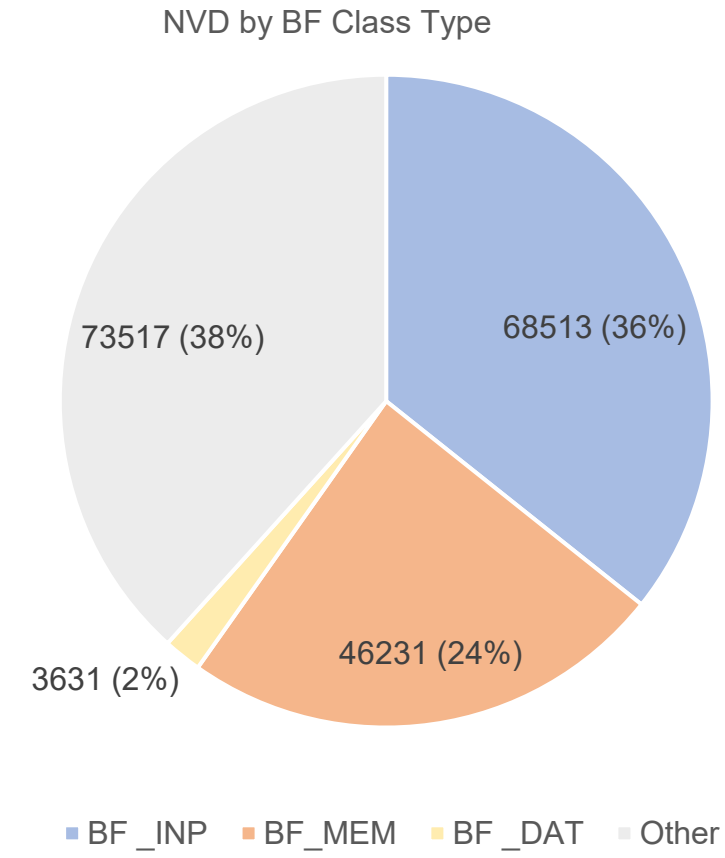
- ✓ NVD description ⟶ BF specification
- ✓ CVSS scores ⟶ BF attributes analytics
- ✓ CWE assignments ⟶ BF security rules

NVD by BF Types

NVD labels 190,000+ CVEs with CWEs

- 60% map to two BF Class Types
  - ✓ 68,000+ ⟶ BF _INP
  - ✓ 46,000+ ⟶ BF _MEM
- Most relate to
  - ✓ Injection ⟵ SQL Injection
  - ✓ Memory Corruption/Disclosure ⟵ Buffer Overflow

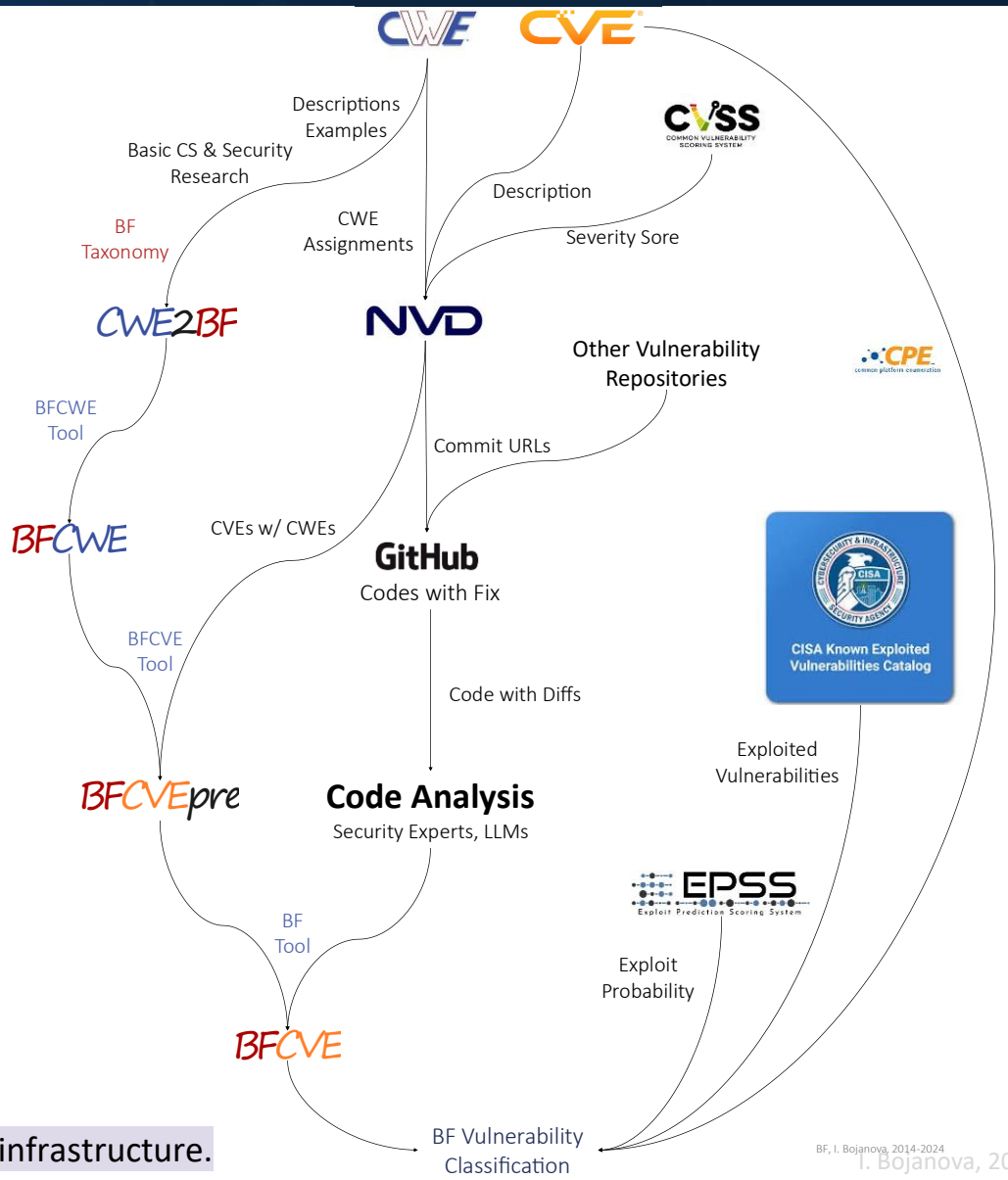
Now it's the time!



Approach

AI-Based Agents

- Analyze ⟵ BF definitions, taxonomy, rules
  - ✓ Narrative descriptions ⟵ NVD, CVE, CWE, KEV, CPE, EPSS
  - ✓ Software ⟵ source code
  - ✓ Hardware code ⟵ firmware, microcode
- Generate ⟵ BF formal language
  - ✓ Vulnerability specifications
  - ✓ Secure coding principles
- Validate/Verify
  - ✓ Toward BF Formal Language



AI-centered vulnerability platform

- ✓ Comprehensively labeled vulnerability datasets
- ✓ AI agents, APIs, UIs, and Apps

NIST Priority: AI-based agents to secure US critical infrastructure.

Impact

NVD–BF (or NVD^{BF}) Platform

- Why NIST?
  - ✓ Team expertise in cybersecurity, FM, AI-based agent
  - ✓ NIST BF inventor, patent in NIST custody
  - ✓ Industry, government, academia partnerships

Outcome

- ✓ Breakthrough vulnerability classifications
- ✓ AI-based agents, APIs, UIs, and Apps

Potential impact

- ✓ Unprecedented cybersecurity analytics capabilities
- ✓ Highly informed cybersecurity R&D innovations
- ✓ Unambiguous communication about cybersecurity
- ✓ Solidifying NIST NVD as authoritative reference

⟶ Establish NIST BF as the standard for specifying cybersecurity vulnerabilities

- Risks if not done
  - ✓ Cybersecurity experts and automated systems will continue to struggle due to lack of precise descriptions of the publicly disclosed vulnerabilities
  - ✓ Parallel efforts—EUVD, OSV, etc.—will eventually do it and gain advantage

NIST Priority: Secure US critical infrastructure.

Potential Collaborators

Expertise

- **NIST Researchers**

Irena Bojanova	—cybersecurity, formal methods—	ITL/775.01
Tim Blattner	—AI models, AI-based agents—	ITL/775.02
Jason Collard	—AI, NLP, formal methods—	ITL/775.04
Jeff Voas	—cybersecurity, digital twin—	ITL/773.03
Rick Kuhn	—combinatorial testing—	ITL/773.02

- **Non-NIST PhD Candidates**

  Purdue University
  Colorado State University
  Idaho National Laboratory
  JHU, Applied Physics Laboratory
  University of West Attica
  University of Quebec

References

[1] Bojanova I (2024) Bugs Framework (BF): Formalizing Cybersecurity Weaknesses and Vulnerabilities. (National Institute of Standards and Technology, Gaithersburg, MD), NIST Special Publication (SP), NIST SP 800-231. https://doi.org/10.6028/NIST.SP.800-231

PATENT PENDING

[2] U.S. Patent Application No. PCT/US2025/038662, Bugs Framework (BF) – A System for Formal Specification of Cybersecurity Weaknesses and Vulnerabilities, Definition of Secure Coding Principles, and Generation of Weakness and Vulnerability Datasets and Vulnerability Classifications. Inventor: Irena Bojanova, NIST.

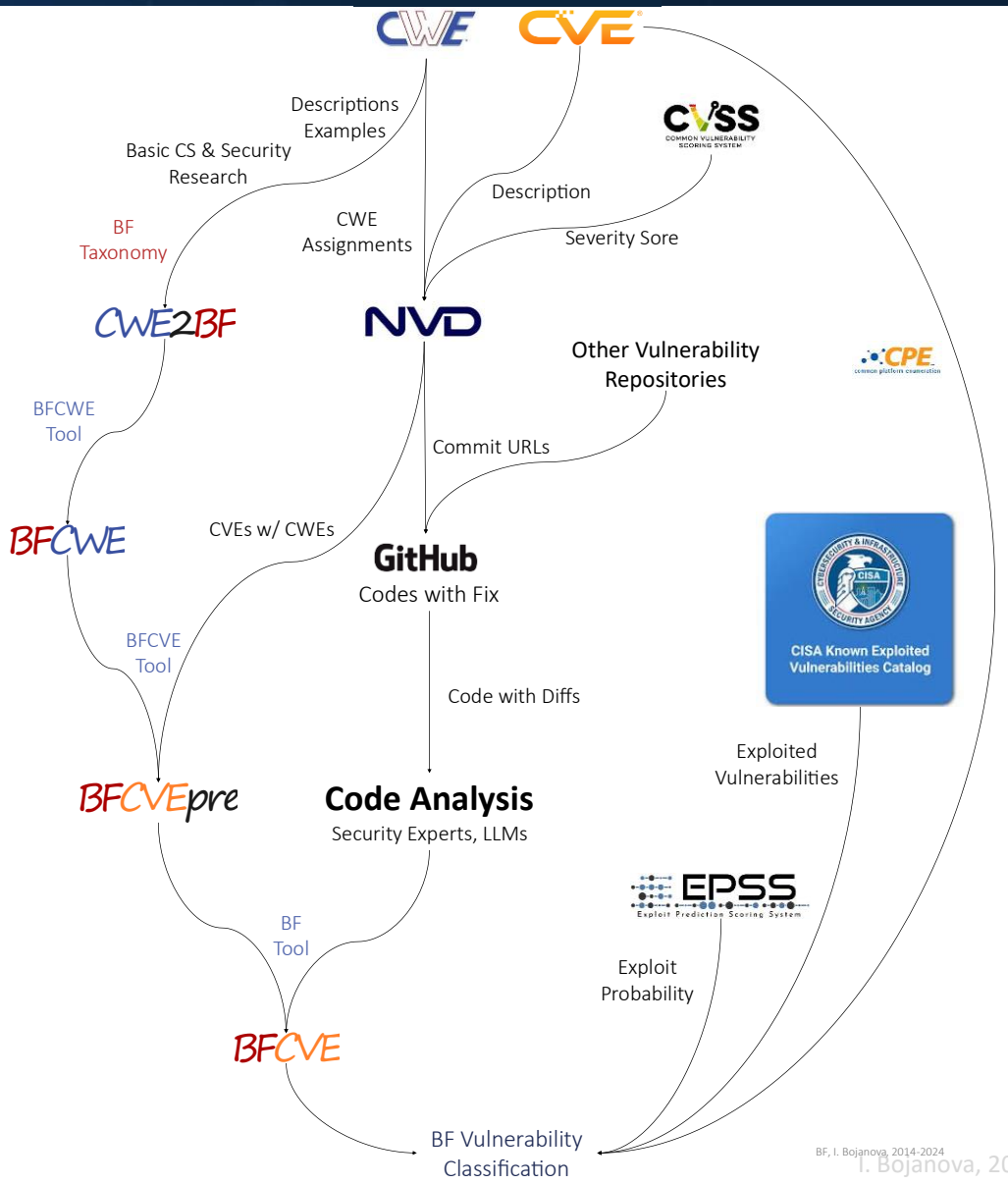
Extras

BF Vulnerability Classifications

- By properties – e.g.,
  - ✓ Root cause – code bug, hardware induced data fault
  - ✓ Propagated error – data fault
  - ✓ Final error – exploit vector
  - ✓ Operation or operand attribute
  - ✓ Combination of BF taxons
- By similarities – e.g.,
  - ✓ Number of underlying weaknesses
  - ✓ Identical chains of weaknesses

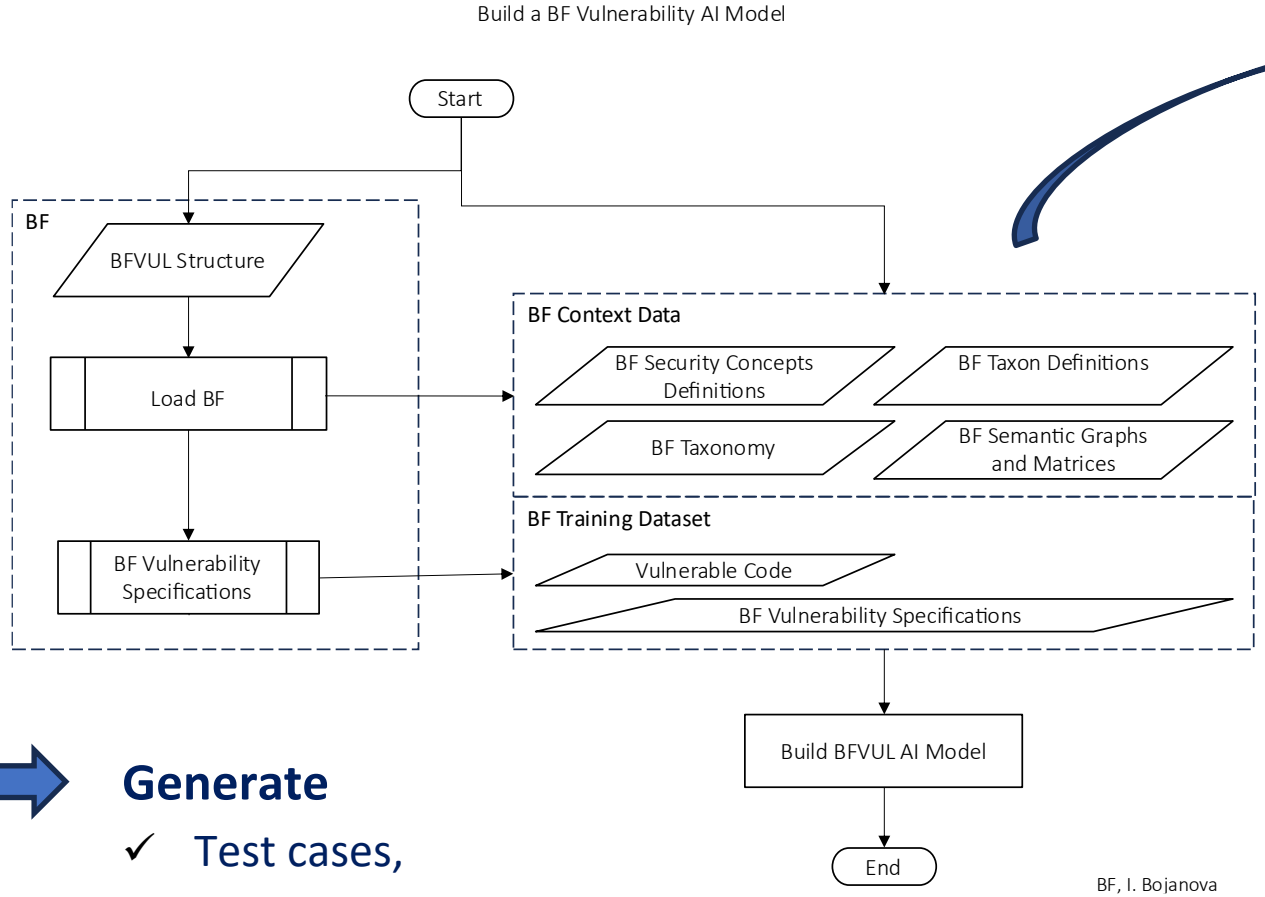
⇒ Deep vulnerability analytics – e.g.,

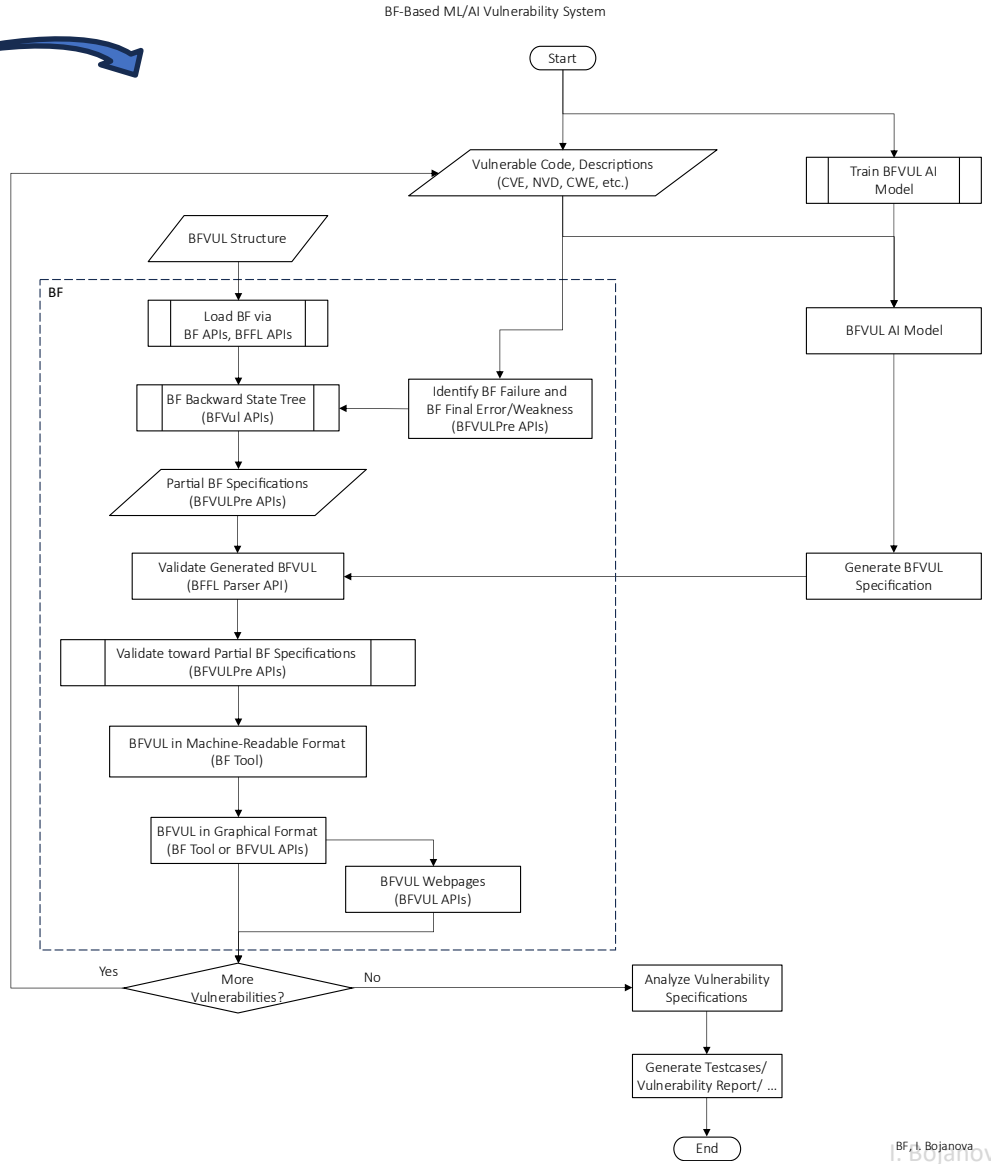
- ✓ Most severe weaknesses/vulnerabilities
- ✓ Most exploited weaknesses/vulnerabilities
- ✓ Systematic input safety issues by vendor/product
- ✓ Systematic memory safety issues by vendor/product



Advance Innovation

BF-Based AI-Powered Systems





Generate

- ✓ Test cases,
- ✓ Vulnerability reports

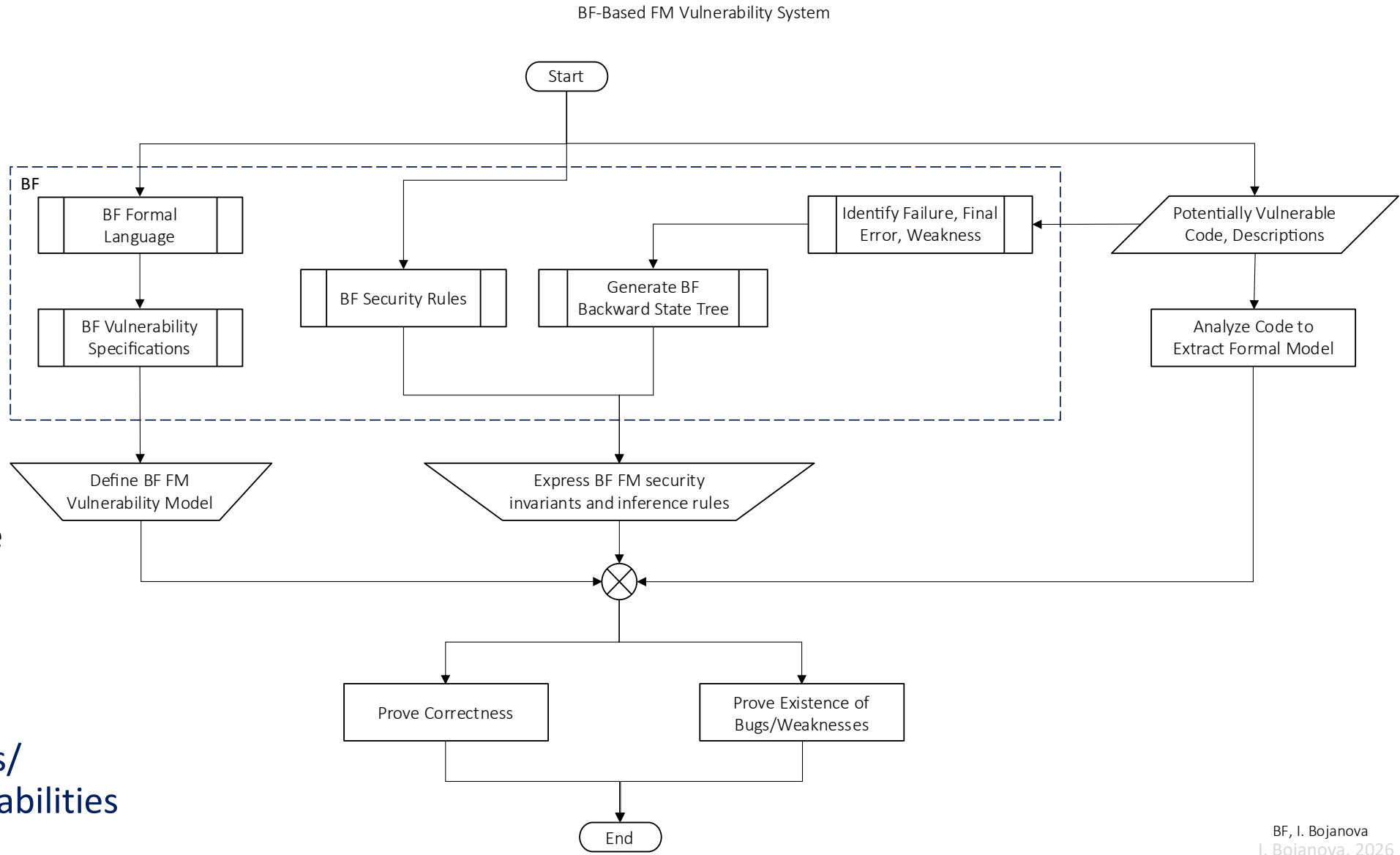
Automate

- ✓ Bug/fault detection and prioritization
- ✓ Vulnerability resolution or mitigation

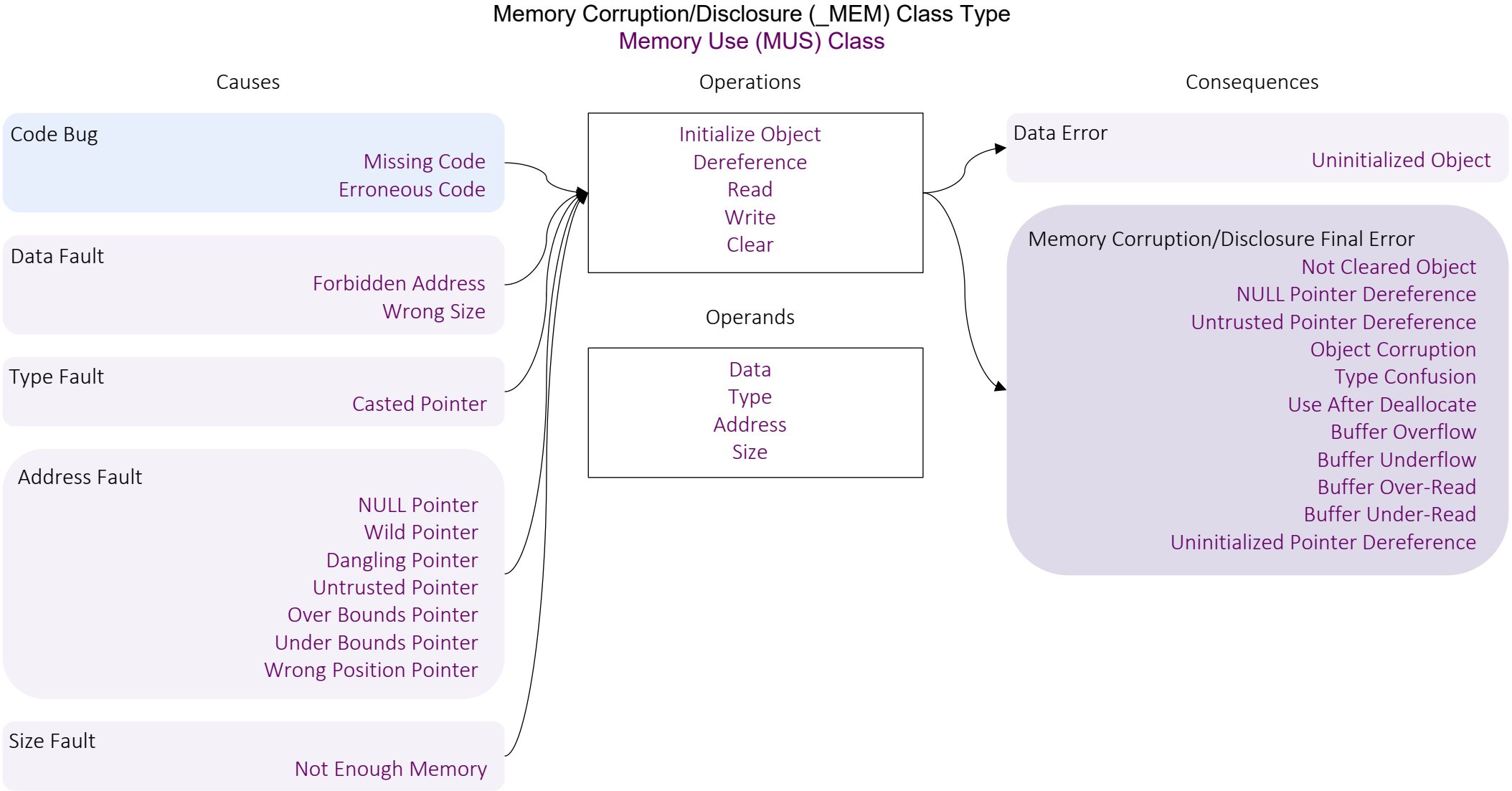
BF-Based FM-Powered Systems

- BF Formal Language
- BF Security Rules
- BF Backward State Tree

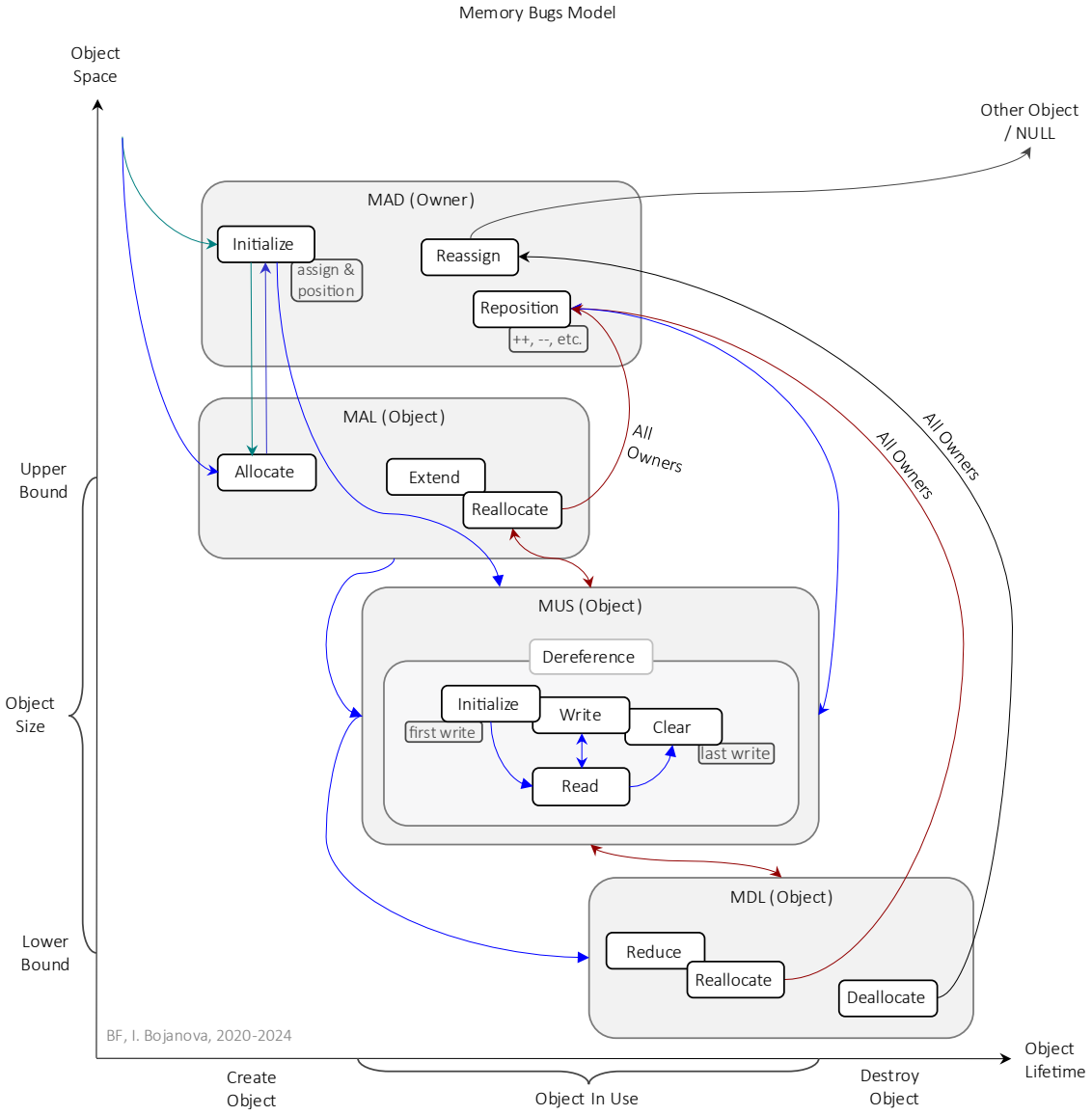
- Prove **correctness**
- Prove **existence** of bugs/ weaknesses and vulnerabilities

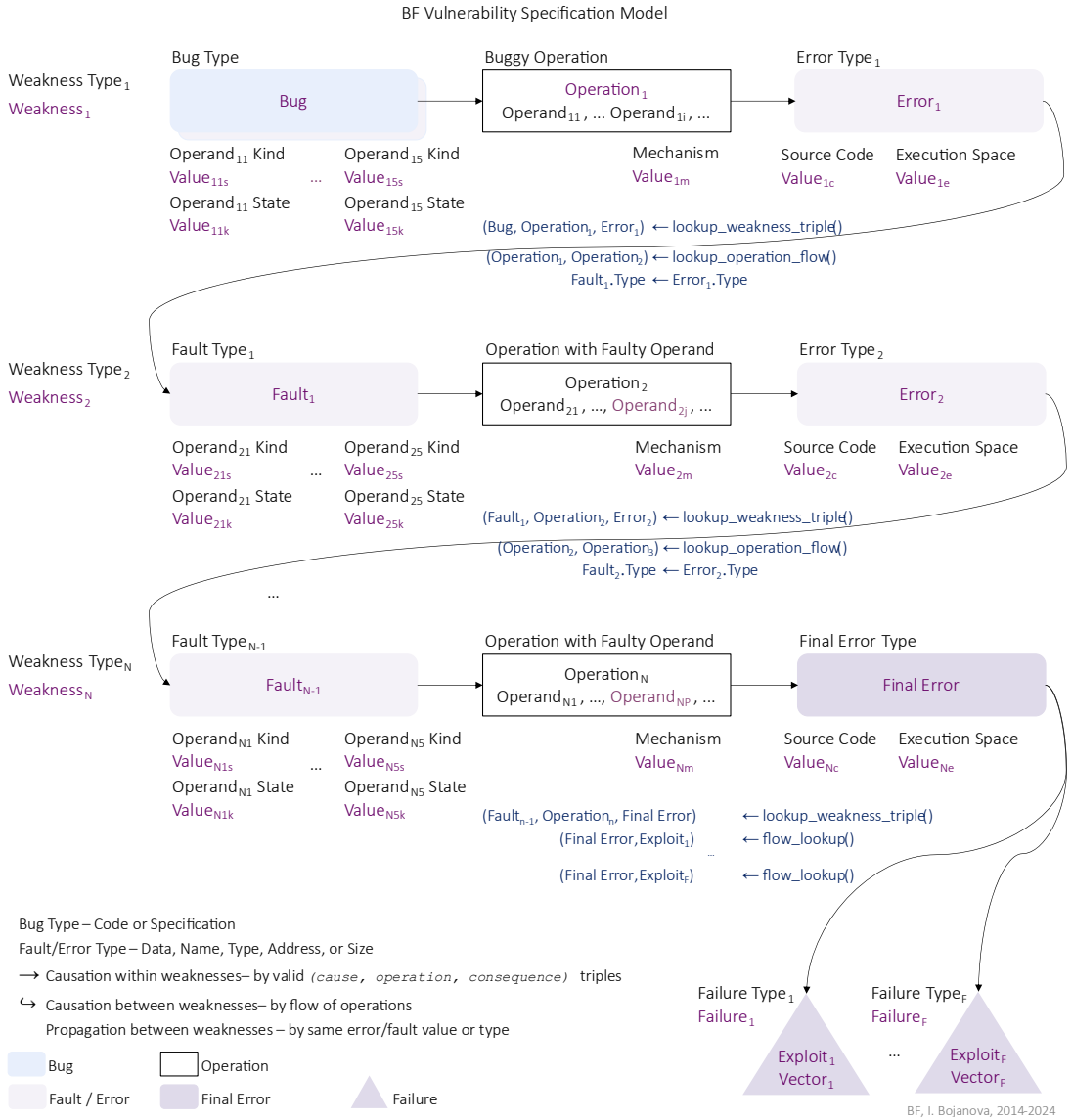


BF Causal Taxonomies



BF Bugs and Specification Models





BF Formal Language

- Specification of cybersecurity weaknesses and vulnerabilities

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

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