Bugs Framework (BF)

Formalizing Software Security Bugs, Weaknesses, and Vulnerabilities

Irena Bojanova, NIST

Motivation

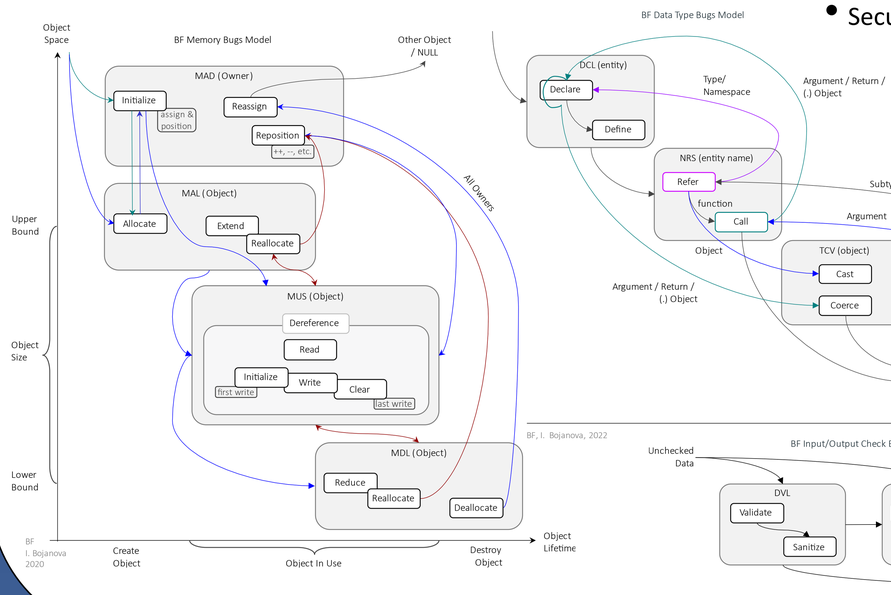
- Crucial need of a formal classification system allowing unambiguous specification of software security bugs and weaknesses, and the vulnerabilities that exploit them.

Objective

- Create bug models, weakness taxonomies, and vulnerability models with causation and propagation rules; and an unambiguous formal weakness/vulnerability specification language.

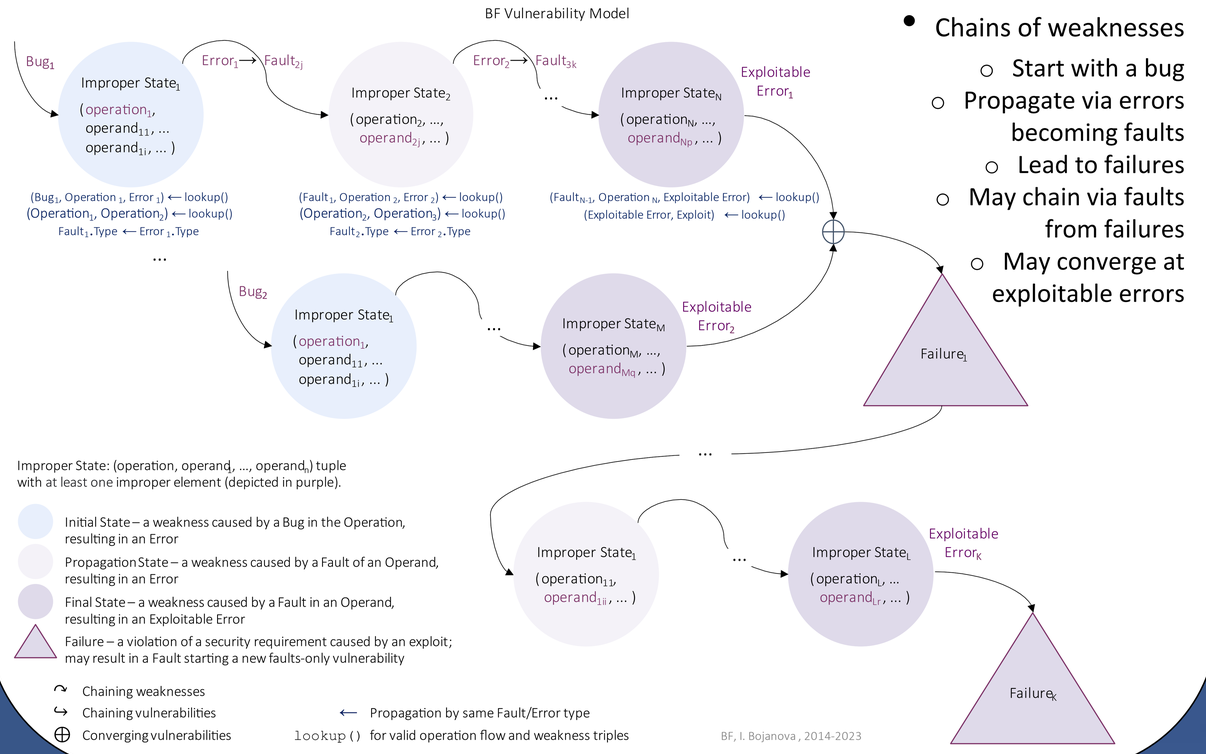
BF Bug Models

- Bug/Fault Type related software execution phases with non-overlapping operations
  - ➢ Causation between weaknesses – by operation flow



- Secure coding principles – e.g.,
  - Input/Output Safety
    - Data Type Safety
      - Memory Safety

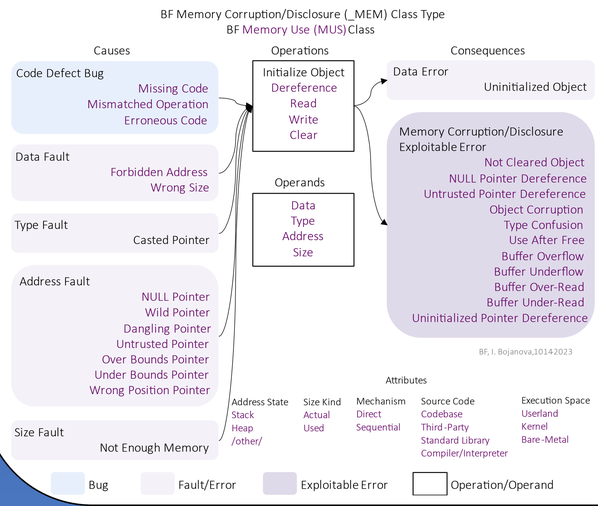
BF Vulnerability Model



- Chains of weaknesses
  - Start with a bug
    - Propagate via errors becoming faults
      - Lead to failures
        - May chain via faults from failures
          - May converge at exploitable errors

BF Weakness Taxonomies

- Weakness Types with valid cause-operation-consequence relations – e.g., the BF MUS class:



- Taxons – types (in black) and values (in purple)
  - Operations and operands
    - Causations and consequences
      - Attributes of operations and operands
  - ➢ Causation within weaknesses – all valid triples (Bug/Fault, Operation, Error/Exploitable Error)
- Structured – cause-operation-consequence
- Complete – no gaps in coverage
- Orthogonal – no overlap by operation
- Context-free – programming language and application domain independent

BF Formal Language

- BF CFG[1] is a G = (V, Σ, R, S) tuple with finite sets of tokens Σ = { α | α is BF taxon} ∪ { ⊕, lookup() }, variables V, rules R = { A ⟶ ω | A ∈ V, ω ∈ (V ∪ Σ)* }, and a predefined start S ∈ V
- BF LL(1)[2] Attribute CFG[3] is BF CFG derived via left-factorization and left recursion elimination

Syntax Rules

S ⟶ Vulnerability Converge_Failure
Vulnerability ⟶ Bug Operation OperAttrs_Error_ExplError
OperAttrs_Error_ExplError ⟶ Oper_Attr OperAttrs_Error_ExplError
| Error Fault OprndAttrs_Operation
| Exploitable_Error
OprndAttrs_Operation ⟶
Operand_Attr OprndAttrs_Operation
| Operation OperAttrs_Error_ExplError
Converge_Failure ⟶
⊕ Vulnerability Converge_Failure
| Exploit NextVulner_Failure
NextVulner_Failure ⟶ Fault Operation OperAttrs_Error_ExplError
| Failure ε

Semantic Rules

(Bug | Fault, Operation, Error | Exploitable Error) ⟵ lookup()
Fault.Type ⟵ Error.Type
Exploit.Type ⟵ Exploitable Error.Type

- BF formal language is the BF LL(1) CFG generated set of all strings derivable from S:

$$L(G) = \{ \alpha \in \Sigma^* \mid S \Rightarrow^* \alpha \}$$

A string starts from S and ends with ε (the empty string) by applying the A ⟶ ω production rules regardless of context.

- ➢ BF LL(1) ⇒ unambiguous BF specifications!
  - Lexis – BF taxons
    - Syntax – BF vulnerability models' flow
      - Semantics – causation & propagation (BF bug models operation flow, valid taxon triples, vulnerability model propagation).

[1] CFG – Context-Free Grammar; [2] LL(1) – Left-to-right Leftmost-derivation One-symbol-lookahead; [3] Attribute CFG – adds attributes, and semantic and predicate functions.

Potential Impact

- Models and a formal language for unambiguous bug, weakness, and vulnerability specifications – for use by professionals, AI algorithms, and systems for training, data generation, and research.