Taxonomy Codebook

February 13, 2026

1 Taxonomy Codebook (v1.0)

Format. Each label is defined with (i) definition, (ii) inclusion cues, and (iii) exclusion cues. Cues are applied only when explicitly supported by title/abstract text.

`VM.SBI` **Security Bug Identification.** *Definition:* Studies that identify or triage security-related bug reports/issues (e.g., predicting whether a report is security-relevant). *Inclusion cues:* "security bug report", "security issue report", "bug report triage", "security-related issue prediction", "vulnerability-related report". *Exclusion cues:* Direct detection in source/binary; patch generation; advisory-to-fix mapping.

`VM.SBI.LB` **Learning-based Bug Identification.** *Definition:* Security bug identification primarily using ML/NLP models over reports or issue text. *Inclusion cues:* "learning-based", "neural", "BERT/Transformer", "text classification" for bug/security reports. *Exclusion cues:* Pure rule-based filtering without learning; code-centric detection.

`VM.VD` **Vulnerability Detection.** *Definition:* Studies whose main output is a detector that flags vulnerabilities in code, binaries, configurations, or running systems. *Inclusion cues:* "detect", "vulnerability detection", "finding vulnerabilities", "vulnerability prediction", "bug finding", "scanner". *Exclusion cues:* Pure severity scoring/ranking (analysis); patch finding; repair.

`VM.VD.GVD` **General-purpose Vulnerability Detection.** *Definition:* Detection methods not restricted to a specific domain (e.g., not limited to smart contracts). *Inclusion cues:* C/C++/Java general code, function-level vulnerability detection, generic datasets (e.g., Big-Vul/Devign). *Exclusion cues:* Explicitly smart-contract-only; explicit layer-scoped threats only.

`VM.VD.GVD.S` **Static Approaches.** *Definition:* Detectors operating without executing the program. *Inclusion cues:* "static analysis", "source code", "bytecode analysis" (non-execution), "program graph", "data/control dependence". *Exclusion cues:* "fuzzing", "dynamic", "runtime tracing", "execution" as primary evidence.

`VM.VD.GVD.D` **Dynamic Approaches.** *Definition:* Detectors relying on execution, runtime traces, or input generation to trigger bugs. *Inclusion cues:* "dynamic analysis", "runtime", "trace", "execution", "fuzzing". *Exclusion cues:* Purely static representations and offline learning without execution.

`VM.VD.GVD.S.GA` **Graph Analysis (non-learning).** *Definition:* Detection driven by program-graph construction and handcrafted/static reasoning rules rather than learned encoders. *Inclusion cues:* "graph-based static analysis", "program dependence graph", "rule-based", "pattern matching", "symbolic constraints". *Exclusion cues:* GNN/Transformer encoders as the primary predictor.

`VM.VD.GVD.S.LB` **Learning-based (static).** *Definition:* Static detection where prediction is driven by learned models over code/graph representations. *Inclusion cues:* "neural", "deep learning", "BERT/Transformer", "GNN", "representation learning". *Exclusion cues:* Pure dynamic testing; non-learning graph rules.

`VM.VD.GVD.S.LB.SEQ` **Sequence Representation.** *Definition:* Learning-based detectors encoding code as sequences (tokens/AST traversal/statement sequences). *Inclusion cues:* "token sequence", "AST sequence", "sequence model", "RNN/GRU/LSTM/Transformer encoder". *Exclusion cues:* Explicit graph message passing as the primary encoder.

`VM.VD.GVD.S.LB.G2S` **Graph-to-Sequence Representation.** *Definition:* Detectors that derive sequences from program graphs (paths/slices/linearizations) and reuse sequence encoders. *Inclusion cues:* "graph-to-sequence", "path-based", "slice sequence", "linearize graph", "CFG/PDG paths". *Exclusion cues:* Direct graph neural encoding as primary driver.

`VM.VD.GVD.S.LB.GRAPH` **Graph Representation.** *Definition:* Detectors whose primary learning view is a program graph directly encoded by graph models (e.g., GNN/message passing). *Inclusion cues:* "GNN", "message passing", "GGNN/GAT/GCN", "program graph encoding", "node/edge aggregation". *Exclusion cues:* Only derived sequences; purely symbolic graph analysis.

`VM.VD.GVD.S.LB.LLM` **LLM-based Detection.** *Definition:* Detectors that use LLM prompting/reasoning (optionally with code graphs) as a primary predictor. *Inclusion cues:* "LLM", "ChatGPT/GPT", "prompt", "in-context learning", "LLM reasoning". *Exclusion cues:* Traditional deep models without LLM; purely static rule tools.

`VM.VD.GVD.D.FUZZ` **Fuzzing-based Detection.** *Definition:* Dynamic detection driven by input generation and runtime oracles. *Inclusion cues:* "fuzzing", "greybox/blackbox", "coverage-guided", "mutations", "oracle", "sanitizer". *Exclusion cues:* Only static scanning without execution.

`VM.VD.SC` **Vulnerability Detection for Smart Contracts.** *Definition:* Detection specialized for smart contracts (e.g., EVM/Solidity/Rust-based chains), including DeFi threats. *Inclusion cues:* "smart contract", "EVM", "Solidity", "DeFi", "reentrancy", "transaction". *Exclusion cues:* Generic software-only detectors without contract context.

`VM.VD.SC.S` **Static Smart-contract Detection.** *Inclusion cues:* "bytecode analysis", "static", "control/data flow" for contracts. *Exclusion cues:* Input-generation/execution as primary.

`VM.VD.SC.D` **Dynamic Smart-contract Detection.** *Inclusion cues:* "fuzzer", "execution", "transaction generation", "runtime monitor". *Exclusion cues:* Purely static analyzers.

`VM.VD.SC.T` **Specific Vulnerability Types (Smart contracts).** *Inclusion cues:* explicit vulnerability names in title (e.g., "reentrancy", "front-running", "price manipulation"). *Exclusion cues:* Broad, non-type-specific scanners.

`VM.VD.LSVD` **Layer-specific Vulnerability Detection.** *Definition:* Detection scoped to a particular system layer (application/middleware/network/system). *Inclusion cues:* "web", "API", "framework", "network stack", "kernel", "compiler". *Exclusion cues:* General detectors without clear layer scoping.

`VM.VD.LSVD.APP` **Application Layer.** *Inclusion cues:* "web app", "REST API", "input validation", "business logic", "XSS/SSRF/SQLi".

`VM.VD.LSVD.MW` **Middleware Layer.** *Inclusion cues:* "framework", "library", "dependency", "distributed middleware", "Android framework".

`VM.VD.LSVD.NET` **Network Layer.** *Inclusion cues:* "protocol", "resolver", "network stack", "DoS/ReDoS".

`VM.VD.LSVD.SYS` **System Layer.** *Inclusion cues:* "kernel", "UAF", "binary", "compiler", "memory safety".

`VM.VA` **Vulnerability Analysis.** *Definition:* Studies focusing on assessing, prioritizing, explaining, or analyzing vulnerability-related artifacts rather than detecting new vulnerabilities. *Inclusion cues:* "severity", "exploitability", "risk ranking", "prioritization", "explanation", "interpretation". *Exclusion cues:* Primary output is a detector; primary output is patch generation.

`VM.VA.AP` **Assessment & Prioritization.** *Inclusion cues:* "CVSS", "severity score", "prioritize", "rank", "risk".

`VM.VA.CI` **Classification & Interpretation.** *Inclusion cues:* "explainable", "interpretation", "rationale", "classification", "attribution".

`VM.VA.AA` **Artifact Analysis.** *Inclusion cues:* "advisory", "asset", "trace", "package", "SBOM", "dependency graph".

`VM.VR` **Vulnerability Repair.** *Definition:* Studies whose main output is a patch/repair method for vulnerabilities. *Inclusion cues:* "repair", "patch generation", "fix", "automated program repair". *Exclusion cues:* Only locating fixes (fix identification) without generating patches.

`VM.VR.CG` **Constraint-guided Repair.** *Inclusion cues:* "constraint", "template", "verification-guided", "spec-guided".

`VM.VR.FZ` **Fuzzing-driven Repair.** *Inclusion cues:* "fuzzing-guided repair", "test generation", "oracle-guided patching".

`VM.VR.PP` **Patch Propagation.** *Inclusion cues:* "propagate port backport", "patch transfer", "multi-branch".

`VM.VR.SCR` **Smart-contract Repair.** *Inclusion cues:* "smart contract repair", "EVM patch", "Solidity fix".

`VM.FI` **Fix Identification.** *Definition:* Studies that identify, localize, or validate security fixes/patches (e.g., mapping advisories to commits). *Inclusion cues:* "fixing commit", "patch identification", "advisory-to-commit", "silent fix", "security patch". *Exclusion cues:* Generating patches (repair); vulnerability detection without fix linkage.

`VM.FI.PF` **Patch Finding.** *Inclusion cues:* "locate fixing commit", "advisory to fix mapping", "rank candidate commits".

`VM.FI.SF` **Silent-fix Identification.** *Inclusion cues:* "silent fix", "undocumented fix", "security-relevant changes" without labels.

`VM.FI.CL` **Commit-level Identification.** *Inclusion cues:* "security commit classification", "commit classification", "fixing commit detection".

`VM.FI.MG` **Multi-granularity Identification.** *Inclusion cues:* "file/function/commit localization", "multi-granularity".