Majestic Journal of Information Technology
Volume 1, 2026, **pp.** 1-21, **Article ID:** CM2601102008

Scoping Review

Smart Contract Security Vulnerabilities and Mitigation Methods: A Scoping Review

Ajay Mahesh Gaur[1, *]

[1]*University of Colorado Denver, Denver, USA*

Article History

Received: 30 July, 2026 Revised: 03 October, 2026 Accepted: 11 October, 2026 Published: 24 October, 2026

Abstract:

Introduction: The research on smart-contract security has gone beyond rule-based and symbolic analysis to runtime defenses and learning-based vulnerability detection, but the evidence is still diverse in terms of vulnerability definitions, validation approaches, and mitigation assertions. The scoping review identifies and maps peer-reviewed research published between 2016 and 2025 and synthesizes the vulnerability families, detection, prevention, verification, and mitigation strategies, validation practices, and evidence gaps.

Methodology: A search in Scopus, Web of Science Core Collection, IEEE Xplore, and Springer Nature Link resulted in 1,200 records: 240 records in Scopus, 469 records in Web of Science, 389 records in IEEE Xplore, and 102 records in Springer Nature Link. A total of 25 evidence sources met the eligibility criteria, comprising 19 primary technical or empirical studies and six secondary reviews, syntheses, or taxonomies.

Results: Structured coding derived seven vulnerability families. External-call/control-flow weaknesses were reported in 23 of the 25 included evidence sources (92%), followed by block/transaction-context weaknesses in 19 sources (76%). Arithmetic/state and logic/semantic/exception weaknesses were each reported in 14 sources (56%). The evidence indicates a temporal shift from symbolic execution, static analysis, fuzzing, and runtime monitoring towards graph-based and deep-learning detection, whereas secure design patterns and exploitability-aware validation remain less comprehensive. Because secondary syntheses may overlap with primary studies represented in the corpus, these frequencies describe document-level literature coverage rather than independent empirical prevalence.

Conclusion: The cross-map therefore supports, rather than proves, a layered assurance approach combining design-time prevention, automated analysis, empirical validation, and runtime protection. The review provides evidence-based vulnerability taxonomies, security-assurance taxonomies of detection, prevention, verification, and mitigation techniques, and a vulnerability-to-security-method cross-map, and reveals the continuing limitations on comparability of benchmarks, cross-platform validation, and real-world exploitability testing.

Keywords: Smart contracts, security vulnerabilities, blockchain security, vulnerability detection, mitigation, scoping review.

1. INTRODUCTION

1.1. Background and Security Context

Smart contracts are computer programs used to encode and execute logic of transactions on distributed ledgers. Their security is not the same as conventional application software because deployed code has the ability to directly manipulate digital assets, is deterministically executed on replicated nodes, and may be monitored publicly. As demonstrated by Chen *et al.* [1], Ethereum insecurity is at the contract, protocol, and environmental levels, which implies that contract-level failure can interact with assumptions at a broader platform level. Similarly, Singh *et al.* [2] describe that formalization is a complicated task since the semantics of blockchain execution, external calls, and irreversible changes of state should be accurately modeled. Such features make security a lifecycle issue and not a narrow code issue. The contract logic, transaction ordering, access control, arithmetic, and exception

*Address correspondence to this author at University of Colorado Denver, Denver, USA; E-mail: ajay.gaur@ucdenver.edu

© 2026 Copyright by the Author.
Licensed as an open access article using a CC BY 4.0 license.

Cite as: A. M. Gaur, "Smart contract security vulnerabilities and mitigation methods: A scoping review," *Maj J Infor Tech.*, vol. 1, p. 1–21, 2026, Article ID: CM2601102008.

handling, as well as the dependence on external state, may introduce vulnerabilities. A sufficient evidence synthesis should thus differentiate between vulnerability mechanisms and the tools and controls that are used to identify them or to control them.

Irreversibility and economic exposure compound the challenge. After deployment of vulnerable code, traditional patching can be limited by governance, upgrade processes, or even the contract design itself. Tolmach *et al.* [3] demonstrate that formal specification and verification have thus become significant research topics, despite the wide variety of methods of verification in their assumptions, expressiveness, and the platforms they can support. Kannengießer *et al.* [4] also name issues related to broader development, such as code visibility, its updateability, testing, verification, and platform-specific limitations, which suggest that security cannot be narrowed down to a type of exploit. Practically, assurance can be a combination of secure design patterns, static and dynamic analysis, formal methods, testing, auditing, and monitoring. The main question of analysis is not about what vulnerabilities are present, but about how well detection, prevention, verification, and mitigation approaches align with the vulnerabilities and what evidence supports their effectiveness.

1.2. Existing Reviews and Research Gap

The reviews that exist can offer useful yet varied scopes of smart-contract security. Chen *et al.* [1] investigate the security of Ethereum systems at a general architectural scale, including vulnerabilities, attacks, and defenses not just in smart contracts. Tolmach *et al.* [3] concentrate on formal specification and verification, which goes into detail about the mathematical assurance but does not systematise the entire list of empirical detection methods. Kannengießer *et al.* [4] concentrate on the issues of development and solutions, such as design patterns and engineering practices. These contributions provide good conceptual underpinnings, but their units of analysis and synthesis goals are different from a scoping map that focuses on the connection between families of vulnerabilities, detection or mitigation practices, and validation evidence.

Surveys of more recent date have expanded the security view but can be further synthesised. Chu *et al.* [5] organise the space into vulnerability data sources, detection, and repair, such as the datasets and performance of the security methods. The mitigation techniques or development solutions have been categorised in other surveys, but it has proven difficult to cross-study them because the vulnerability labels, benchmarks, platforms, and evaluation metrics are not standardised. This forms an expedient discrepancy between the array of detection strategies on offer and the knowledge of which vulnerability groups are recurrently covered, which controls are enforced on them, and in which areas validation is feeble. A scoping approach is appropriate as it can map heterogeneous primary tools, empirical tests, taxonomies, and secondary syntheses without forcing them to a single effect-size model. This review is therefore valuable as a systematized cross-map of vulnerabilities and families of security-assurance methods, validation strategies, and temporal development within a particular 2016-2025 evidence window.

1.3. Purpose and Contribution

This review will focus on mapping peer-reviewed evidence on vulnerabilities of smart-contract security, and the technical mechanisms of identifying, preventing, verifying, or mitigating vulnerabilities. There are four contributions of the review. It then generates a seven-family vulnerability taxonomy, which is the standardisation of inconsistent terms into analytically similar groups. Second, it separates detection, prevention, verification, and mitigation strategies by technical mechanism, as opposed to considering auditing, formal verification, static analysis, and machine learning as equivalent forms of controls. Third, it associates the vulnerability families with the security strategies that are supported by the evidence found in the documents and records the kind of validation used. Fourth, it explores temporal and methodological trends to determine the areas in which the literature has been developed and those in which the evidence is still concentrated. Collectively, these contributions offer a productive paradigm to compare security strategies while maintaining the heterogeneity anticipated of a scoping review.

1.4. Research Questions

RQ1: Which vulnerability families are most frequently reported in peer-reviewed smart-contract security evidence sources published from 2016 to 2025?

RQ2: Which detection, prevention, verification, and mitigation approaches are used for each vulnerability family, and what forms of validation support them?

RQ3: How have smart-contract security methods and research emphases evolved across the 2016–2025 period?

RQ4: What evidence gaps remain in vulnerability coverage, validation, mitigation, and generalizability?

2. METHODOLOGY

2.1. Review Design and Reporting Framework

A scoping review design was chosen due to the nature of the research questions that necessitate the mapping of the breadth, characteristics, and relationships of heterogeneous technical literature, as opposed to estimating one intervention effect. The review follows the staged framework proposed by Arksey and O'Malley [6], incorporates the updated conduct guidance of Peters *et al.* [7], and is reported according to the PRISMA-ScR guidance outlined by Tricco *et al.* [8]. Munn *et al.* [9] differentiate between scoping and traditional systematic reviews in cases where the purpose is to clarify the concepts, illustrate the types of evidence used, analyse the process of conducting research, and define gaps in knowledge. These functions align with the current review since the evidence consists of both static and dynamic analysis, formal methods, runtime defenses, machine-learning models, taxonomies, empirical benchmark studies, and secondary technical syntheses. No prospective protocol was registered, but the eligibility criteria, search fields, database syntax, charting variables, and synthesis procedures are reported explicitly to facilitate reproducibility. An overview PRISMA-ScR reporting crosswalk is provided in Appendix **E**.

2.2. Eligibility Criteria

Pre-synthesis criteria were established to ensure that the substantive corpus was limited to smart-contract security as opposed to blockchain applications in general. To qualify as evidence sources, the source had to be published between 2016 and 2025, be a peer-reviewed journal article or a conference proceeding, contain readable full text to chart the data, and discuss either smart-contract vulnerabilities, exploit mechanisms, vulnerability detection, formal verification, security analysis, auditing, runtime protection, secure design patterns, or a closely related mitigation. Primary technical studies, as well as directly relevant secondary technical syntheses, were eligible since evidence mapping, and not effect estimation, was the aim. In this review, primary studies are considered as a subset of empirical, experimental, or tool-development research, but systematic reviews, technical reviews, taxonomies, and meta-analytic comparisons are considered as secondary evidence sources. Papers that involved smart-contract security incidental to the primary topic were filtered out by general blockchain, Internet of Things, healthcare, agriculture, finance, or network-security topics. Duplicates or outdated versions, non-peer-reviewed preprints, editorials, blogs, and nontechnical discussions of governance were also filtered out. These criteria are operationalised in Table **1**.

2.3. Information Sources, Search Date, and Search Fields

Searches were conducted on 15 September 2026 in Scopus, Web of Science Core Collection, IEEE Xplore, and Springer Nature Link. The fixed publication window was 2016-2025 to give the full ten years of publication and to avoid 2026 publications that are not fully indexed. Search syntax was modified to the indexing structure of each database, but maintained the same conceptual blocks: smart contracts, security or vulnerability terminology, and vulnerability-specific or mitigation-specific terms. Title and Title-Abstract-Keyword fields were used in Scopus, separate Title and Topic fields in Web of Science, Document Title constraints in the core smart-contract security concept, and an extra vulnerability/mitigation block in IEEE Xplore and Advanced Search with Title and Keywords fields in Springer Nature Link. Evidence of the search interface is presented in Appendices **A**-**D**, and Table **2** summarises the confirmed counts of the database.

Table 1. Operational eligibility criteria.

Criterion	Include	Exclude
Publication period	2016-2025 inclusive	Before 2016 or after 2025
Language	English	Non-English
Publication status	Peer-reviewed journal article or conference proceeding	Editorial, blog, industry-only item, or non-peer-reviewed preprint
Technical focus	Direct analysis of smart-contract vulnerabilities, exploits, detection, verification, auditing, runtime protection, secure patterns, or mitigation	General blockchain application or cybersecurity study without substantive smart-contract security analysis
Evidence type	Primary empirical/tool study or directly relevant technical review/taxonomy	Nontechnical commentary or unrelated methodological literature
Full text	Readable full text available for charting	Insufficient full text for reliable extraction
Version control	Most complete peer-reviewed version retained.	Duplicate or superseded conference/preprint version when a fuller peer-reviewed version was available

Note: *Eligibility was applied to the substantive security corpus; methodological references used to justify the review design were not counted as included evidence sources.*

Table 2. Database search summary and verified identification counts.

Database	Search Fields	Period	Language Handling	Records
Scopus	TITLE; TITLE-ABS-KEY	2016-2025	English filter applied	240
Web of Science Core Collection	Title; Topic	2016-01-01 to 2025-12-31	English filter applied	469
IEEE Xplore	Document Title plus vulnerability/mitigation terms	Eligibility window 2016-2025; interface displayed 2018-2025 because no matching 2016-2017 records were returned	English applied at eligibility screening	389
Springer Nature Link	Advanced Search: Title; Keywords	2016-2025	English filter applied	102
Total	-	-	-	1,200

Note: *Counts reflect the documented search interfaces on the search date. Database coverage and indexing change over time, so the counts are date-specific.*

2.4. Database-Specific Search Strategies

The search terms have been maintained narrow enough to limit the blockchain noise but include vulnerability and mitigation terms. Precise syntax is replicated below since field codes and phrase processing vary across platforms. There was no document-type limitation in the initial search in Scopus or Web of Science; the publication status was evaluated during the eligibility screening. Type of content at Springer was also not restricted at identification. Appendices **A**-**D** should be read along with the exact search strings and the corresponding result-page evidence.

2.4.1. Scopus

```
    TITLE(
  "smart contract*" AND (securit* OR vulnerab* OR exploit*)
)
AND TITLE-ABS-KEY(
  reentrancy OR "integer overflow" OR "integer underflow" OR "access control"
  OR "transaction ordering" OR "front running" OR "timestamp dependence"
  OR "denial of service" OR "logic flaw*" OR "security flaw*" OR mitigat*
  OR countermeasure* OR prevent* OR detect* OR audit* OR "formal verification"
  OR "static analysis" OR "dynamic analysis" OR "secure coding"
)
AND PUBYEAR > 2015
AND PUBYEAR < 2026
AND LIMIT-TO(LANGUAGE, "English")
```

The Scopus query returned 240 records. Appendix **A** reproduces the final result page showing the query and count.

2.4.2. Web of Science Core Collection

```
    Title field:
"smart contract*" AND (securit* OR vulnerab* OR exploit*)

AND Topic field:
reentrancy OR "integer overflow" OR "integer underflow" OR "access control"
OR "transaction ordering" OR "front running" OR "front-running"
OR "timestamp dependence" OR "denial of service" OR "logic flaw*."
OR "security flaw*" OR mitigat* OR countermeasure* OR "vulnerability detection"
OR "security audit*" OR "formal verification" OR "static analysis"
OR "dynamic analysis" OR "secure coding" OR "security tool*."

Publication date: 2016-01-01 to 2025-12-31; Language: English
```

The Web of Science search returned 469 records after the English and complete 2016–2025 date filters were applied. Appendix **B** contains the fielded query and final result evidence.

2.4.3. IEEE Xplore

```
   (
 "Document Title": "smart contract*"
 AND (
  "Document Title":securit* OR "Document Title":vulnerab*
  OR "Document Title": exploit*
 )
)
AND (
 reentrancy OR "integer overflow" OR "access control" OR "transaction ordering"
 OR "front running" OR "timestamp dependence" OR mitigat* OR countermeasure*
 OR detect* OR audit* OR "formal verification" OR "static analysis"
 OR "dynamic analysis" OR "secure coding"
)
```

The predefined eligibility period was 2016–2025. IEEE Xplore displayed the applied result range as 2018–2025 because the final query returned no matching records for 2016 or 2017. The resulting set contained 389 records, as shown in Appendix **C**.

2.4.4. Springer Nature Link

Title field:
"smart contract" AND (security OR vulnerability OR vulnerabilities OR exploit)

Keywords field:
reentrancy OR "integer overflow" OR "integer underflow" OR "access control"
OR "transaction ordering" OR "front running" OR "front-running"
OR "timestamp dependence" OR "denial of service" OR "security flaw"
OR mitigation OR countermeasure OR "vulnerability detection" OR "security audit"
OR "formal verification" OR "static analysis" OR "dynamic analysis"
OR "secure coding"

Filters: 2016-2025; English

The Springer Nature Link Advanced Search returned 102 records. Appendix **D** reproduces the final result interface with the 2016–2025 and English filters visible.

2.5. Screening and Study Selection

After consolidation, 1,200 records were found using the four database searches: Scopus (n = 240), Web of Science Core Collection (n = 469), IEEE Xplore (n = 389), and Springer Nature Link (n = 102); after removing 210 duplicate records, 990 records passed through title and abstract screening. The total number of records at this stage was 720 records, which failed to meet the technical focus and eligibility criteria in Table **1**, and 270 reports were retrieved. Fifteen reports could not be retrieved, and 255 full-text reports were assessed for eligibility.

Full-text assessment excluded 230 reports: general blockchain or cybersecurity studies without substantive smart-contract security analysis (n = 64); studies in which smart-contract security was incidental to the main focus (n = 48); non-peer-reviewed publications, editorials, blogs, or preprints (n = 39); duplicate or superseded publication versions identified at the full-text stage (n = 27); publications outside the 2016–2025 period or not published in English (n = 22); reports with insufficient information for the required data extraction (n = 17); and nontechnical governance or application-focused studies outside the review scope (n = 13). These exclusions totalled 230, leaving 25 evidence sources in the final synthesis. Two reviewers independently screened titles and abstracts against the predetermined eligibility criteria and independently assessed the retrieved full texts. Any record considered potentially relevant by either reviewer was retained for full-text assessment. Disagreements at either stage were resolved through discussion and consensus before the final eligibility decision was recorded. No formal inter-rater reliability coefficient was calculated. A decision was made on eligibility using the operational criteria in Table **1**, and records that were uncertain were taken forward to the full-text assessment instead of being eliminated based on the wording of the title. The entire numerical reconciliation of the selection process is shown in Fig. (**1**).

2.6. Data Charting

A structured charting form was used to extract bibliographic and technical information from each included evidence source. The chart included the year of publication, evidence type, blockchain platform, programming language or representation, type of vulnerability, detection, prevention, verification, or mitigation mechanism, validation mechanism, sample or benchmark details, and significant finding. Primary empirical tools and experiments were coded as primary evidence, whereas systematic reviews, technical reviews, taxonomies, and meta-analytic comparisons were coded as secondary evidence. This distinction prevents synthesis-level observations from being treated as independent attack observations. The categories in data-charting were merged when they used semantically similar labels to describe the same concept, *e.g.,* transaction-ordering dependence and front-running or arithmetic overflow and underflow. The final extraction matrix is given in Table **3**.

2.7. Coding, Thematic Synthesis, and Frequency Analysis

The synthesis used directed qualitative coding followed by cross-source comparison. The labels of vulnerability were coded on the evidence-source level and subsequently aggregated into seven higher-order families such as external-call / control-flow, arithmetic / state, block/transaction context, authorization / access control, resource / availability, lifecycle / fund-safety, and logic/semantic/exception handling. Security approaches were coded into distinct detection, prevention, verification, and mitigation families, including static analysis, symbolic / formal methods, fuzzing and runtime monitoring, machine / deep learning, secure coding and design patterns, multi-tool auditing and benchmarking, and taxonomy/evidence mapping. Document-level reporting frequency was then calculated for each vulnerability family. Each included publication could contribute a maximum of one count to a given vulnerability family. Source type was retained during coding to limit overlap in interpretation: the six secondary evidence sources were used to harmonise terminology, support taxonomy development, and contextualise recurring vulnerability-method relationships, but they were not treated as independent empirical replications of primary findings. Primary studies remained the principal basis for interpreting tool performance, exploitability, validation, and operational effectiveness. Consequently, a vulnerability reported in both a primary study and a secondary synthesis could contribute to document-level coverage, but this was interpreted as convergence in the literature rather than two independent empirical observations. The resulting frequencies

therefore represent coverage across the 25 evidence sources rather than prevalence, effect size, or independent attack incidence.

2.8. Quality and evidence considerations

No formal risk-of-bias scoring was done since the evidence included in this study consists of incompatible designs, such as software tools, benchmark experiments, runtime defenses, empirical chain analyses, taxonomies, systematic reviews, and meta-analytic comparisons. Following the mapping objective of a scoping review, methodological heterogeneity was managed by mapping evidence type, validation method, dataset or benchmark, and evidence level, instead of statistically combining results. Special care was taken in the interpretation of accuracy or precision values since benchmark composition, vulnerability labelling, source-code availability, compiler version, and platform coverage vary across sources. Claims of comparative effectiveness are therefore interpreted within the validation conditions reported by each publication rather than as universal performance rankings.

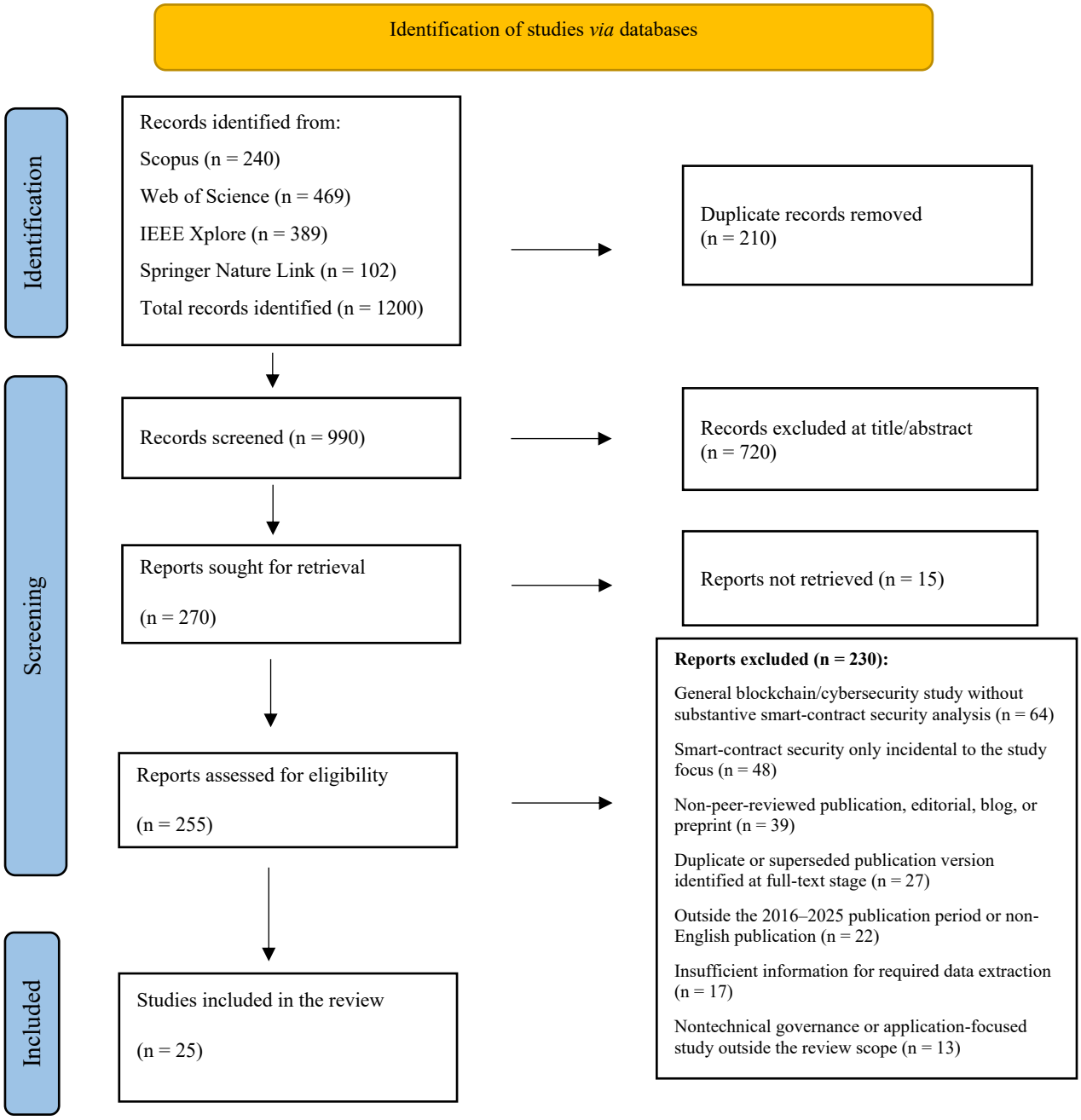


Fig. (1). PRISMA-ScR flow diagram for study identification and selection.

Table 3. Characteristics and evidence extracted from the 25 included evidence sources.

Evidence Source	Design / Evidence	Platform / Language	Vulnerability Focus	Detection / Prevention / Verification / Mitigation Approach	Validation Basis	Evidence Level	Principal Contribution
Luu *et al.* [10].	Primary tool / empirical	Ethereum; EVM / Solidity	Transaction ordering, timestamp, exceptions, reentrancy	Symbolic execution (Oyente) and protocol-level mitigation	19,366 contracts; 8,833 flagged as potentially vulnerable	Primary empirical/tool evidence	Early large-scale evidence that symbolic execution can expose multiple contract-level weaknesses.
Atzei *et al.* [11].	Technical survey/taxonomy	Ethereum; Solidity / EVM	External calls, gas, exceptions, type casts, reentrancy, randomness, time, fund loss	Attack taxonomy and defensive reasoning	Attack cases and platform-layer analysis	Secondary evidence/taxonomy	Shows that vulnerabilities originate across Solidity, EVM, and blockchain interaction layers.
Tikhomirov *et al.* [12].	Primary static-analysis tool	Ethereum; Solidity	Unchecked calls, reentrancy, tx. origin, arithmetic, timestamp, gas, visibility	Pattern-based source analysis (SmartCheck)	Real-world contracts and tool comparison	Primary empirical/tool evidence	Provides extensible source-level detection using syntax transformation and rule patterns.
Jiang *et al.* [13].	Primary dynamic-analysis tool	Ethereum; EVM	Gasless send, exception disorder, reentrancy, timestamp/block number, delegatecall, frozen Ether	Fuzzing with vulnerability-specific oracles (ContractFuzzer)	6,991 contracts; more than 459 vulnerabilities reported	Primary experimental evidence	Demonstrates scalable dynamic testing for vulnerability classes that depend on execution behavior.
Tsankov *et al.* [14].	Primary static/security analyser	Ethereum; EVM bytecode	Reentrancy, transaction dependence, restricted writes/transfers, exceptions, locked Ether	Dependency graph plus compliance/violation patterns (Securify)	Large contract set and manual validation	Primary empirical/tool evidence	Distinguishes evidence for compliance from evidence for definite security-pattern violations.
Nikolic *et al.* [15].	Primary symbolic / trace analysis	Ethereum bytecode	Greedy, prodigal, and suicidal trace vulnerabilities	Interprocedural symbolic analysis with concrete validation (MAIAN)	Nearly one million contracts; manually checked sample with high true-positive rate	Primary empirical/tool evidence	Extends analysis from local code defects to stateful multi-transaction trace properties.
Krupp & Rossow [16].	Primary exploit-generation study	Ethereum bytecode	Unsafe CALL, CALLCODE/DELEGATECALL, SELFDESTRUCT and unrestricted value flow	Symbolic exploit generation (teEther)	38,757 unique contracts; working exploits generated for 815	Primary experimental/runtime evidence	Links vulnerability discovery to concrete exploitability rather than flags alone.
Rodler *et al.* [17].	Runtime protection study	Ethereum	Reentrancy	Dynamic taint tracking and runtime monitoring (Sereum)	Real-chain replay; reported low false-positive rate and low overhead	Primary experimental/runtime evidence	Provides post-deployment reentrancy protection without requiring source-code changes.

Feist *et al.* [18].	Static-analysis framework	Ethereum; Solidity	Multiple Solidity defects, with detailed reentrancy evaluation	SlithIR, data-flow and taint analysis (Slither)	Real contracts; manual review of high-use contracts	Primary empirical/tool evidence	Combines speed, extensibility, and low-level analyses for security auditing.
Frank *et al.* [19]	Bounded model checking	Ethereum bytecode	Suicidal contracts and unrestricted value extraction	Precise EVM modelling and bounded model checking (ETHBMC)	Approximately 2.2 million accounts; 5,905 valid triggering inputs	Primary computational/experimental evidence	Shows that precise bounded reasoning can validate exploitable states at blockchain scale.
Zhuang *et al.* [20]	Machine-learning study	Ethereum; Solidity graph representation	Reentrancy, timestamp dependence, infinite loop	Graph neural network with temporal message propagation	Comparative experiments on labelled smart-contract functions	Primary computational/experimental evidence	Demonstrates graph-based representation learning for structural vulnerability patterns.
Durieux *et al.* [21].	Comparative empirical study	Ethereum; Solidity	Multiple vulnerability classes across nine tools	Cross-tool benchmark and agreement analysis	69 annotated vulnerable contracts plus 47,587 real contracts	Primary comparative empirical evidence	Shows substantial inconsistency in coverage, execution success, and findings among analysts.
Perez & Livshits [22].	Empirical exploitability study	Ethereum on-chain data	Vulnerabilities reported by six analysis projects	On-chain exploitation verification	23,327 vulnerable contracts; 1.98% observed as exploited	Primary empirical/on-chain evidence	Demonstrates that vulnerability detection should not be equated automatically with actual exploitation.
Huang *et al.* [23].	Machine-learning study	Ethereum; Solidity	Arithmetic, reentrancy, unknown-address behavior	Multi-task vulnerability classification	Precision, recall, and F1 comparison with analysis baselines	Primary computational/experimental evidence	Uses shared representation learning to identify multiple vulnerability types jointly.
Zhang *et al.* [24].	Deep-learning study	Ethereum; Solidity	Callstack, overflow/underflow, reentrancy, timestamp, ordering/loop issues	Hybrid CNN-BiGRU model (CBGRU)	SmartBugs Dataset-Wild; comparative classification metrics	Primary computational/experimental evidence	Integrates local and sequential code features for multi-class vulnerability detection.
Ashizawa *et al.* [25].	Machine-learning static analysis	Ethereum; Solidity / EVM	Seven types, including reentrancy, time dependency, gas, visibility, overflow/underflow	Contract-wide code embeddings (Eth2Vec)	5,000 files / 95,152 contracts; 10-fold validation; average precision 77.0%	Primary computational/experimental evidence	Improves robustness to code rewrites by learning contract-wide representations.
Kushwaha *et al.* [26].	Systematic review	Ethereum; Solidity	Broad vulnerability and attack coverage	Review of detection tools, attacks, and countermeasures	Systematic technical synthesis	Secondary evidence / systematic review	Maps recurrent Ethereum vulnerability classes to available detection and prevention mechanisms.

Zhou *et al.* [27].	Technical review	Ethereum	Broad smart-contract weaknesses	Vulnerability, countermeasure, and tool-support synthesis	Survey comparison of techniques and tools	Secondary evidence / technical review	Connects vulnerability categories to countermeasures and practical tool support.
Ma *et al.* [28]	Graph deep-learning study	Ethereum; Solidity	Reentrancy, timestamp, overflow, underflow	Hierarchical graph attention network (HGAT)	SmartBugs data; 7,018 contracts and comparative experiments	Primary computational/experimental evidence	Combines hierarchical structural representations for improved multi-class detection.
Tang *et al.* [29].	Deep-learning study	Ethereum; Solidity	Reentrancy, timestamp, unhandled exception, tx. origin	Lightning Cat using optimized CodeBERT / neural models	SolidiFI benchmark; 9,369 contracts; best reported F1 93.53%	Primary computational/experimental evidence	Shows the benefit of semantic pretrained representations for vulnerability classification.
Deng *et al.* [30]	Multimodal deep-learning study	Ethereum; source, opcode, control-flow	Arithmetic, reentrancy, transaction ordering, Ether locking	Multimodal feature fusion	Per-class accuracy reported around 89.5%-94.8%	Primary computational/experimental evidence	Combines complementary code modalities to reduce dependence on a single representation.
Xu *et al.* [31]	Heterogeneous-graph deep learning	Ethereum; Solidity	Access control, arithmetic, DoS, front-running, reentrancy, time, unchecked low-level calls	MVD-HG using AST, CFG, DFG and multi-granularity learning	Contract- and line-level experiments with repeated runs	Primary computational/experimental evidence	Broadens detection across vulnerability classes while adding line-level localization.
Vidal *et al.* [32].	Taxonomy/evidence synthesis	Smart-contract security ecosystem	Broad vulnerability landscape	Open hierarchical taxonomy (OpenSCV)	481 collected vulnerability entries and 77 detection tools considered	Secondary evidence / taxonomy and evidence synthesis	Provides a normalised vocabulary for heterogeneous vulnerability classifications.
Bresil *et al.* [33].	Comparative review/meta-analysis	Smart contracts; mainly Ethereum	Deep-learning detection, especially reentrancy and timestamp dependence	Comparative analysis of deep-learning architectures	Cross-study performance synthesis	Secondary evidence/review and meta-analysis	Shows that representation and architecture choices materially shape reported detection performance.
Azimi *et al.* [34].	Systematic review/design patterns	Smart contracts	Security-relevant design flaws and known vulnerabilities	Security design-pattern mapping	144 patterns; 36 security patterns; five patterns directly mapped to known vulnerabilities	Secondary evidence / systematic review	Reveals a gap between the breadth of known vulnerabilities and directly targeted security patterns.

Note: *Secondary reviews and taxonomies are retained as secondary evidence and are distinguished from primary tool or empirical studies to avoid treating synthesis-level observations as independent attack incidence.*

3. RESULTS

3.1. Study Selection and Corpus Profile

This section presents the study-selection process and the profile of the final evidence corpus.

Search interfaces found 1,200 records: 240 in Scopus, 469 in Web of Science Core Collection, 389 in IEEE Xplore and 102 in Springer Nature Link. Following the deletion of 210 duplicate records, 990 records were screened by title and abstract, with 720 being eliminated. Among the 270 reports requested to be retrieved, 15 were not retrieved, and 255 full-text reports were available to assess eligibility. An additional 230 reports were filtered out based on the reasons indicated in Fig. (**1**) to arrive at 25 evidence sources that were included. The final corpus spans 2016–2025 and comprises 19 primary technical or empirical studies and six secondary reviews, syntheses, or taxonomies. It is highly focused on Ethereum, Solidity, and EVM-level analysis. It encompasses primary vulnerability-detection tools, symbolic and formal analysers, fuzzing, and runtime-defense systems, machine- and deep-learning models, empirical benchmark comparisons, vulnerability taxonomies, and secondary syntheses. Table **3** presents the cross-source extraction matrix, and Fig. (**2**) presents the temporal distribution. The distribution indicates an early focus on static, symbolic, and exploit-oriented analysis followed by growth in learning-based approaches.

3.2. Vulnerability Taxonomy and Reporting Frequency

The coding exercise resulted in seven vulnerability families. The most commonly reported category was the external-call/control-flow family, represented in 23 of 25 included evidence sources (92%). This type comprises reentrancy, unsafe external calls, delegatecall abuse, call stack behavior, and unchecked low-level interactions. Reentrancy is particularly intractable in the evidence base: it is discussed by early symbolic and rule-based analysers, dynamic fuzzers, runtime protection, machine-learning classifiers, and subsequent heterogeneous-graph models. Luu *et al.* [10] illustrate the initial symbolic-execution strategy, and Rodler *et al.* [17] move the control point to runtime defense. Xu *et al.* [31] demonstrate that multi-class, line-level graph learning continues with the same family as a target. Its persistence through methodological generations suggests that the innovation of tools has transformed more quickly than it has eradicated the underlying vulnerability group.

Block/transaction-context weaknesses were reported in 19 evidence sources (76%), including dependence on timestamps, transaction ordering, front-running, weak randomness, and block-state assumptions. Atzei *et al.* [11] place these issues in the context of language, EVM, and blockchain interaction layers, which contributes to the inability to remove them with local source-code inspection only. ContractFuzzer implements a number of context-specific behaviors in the form of dynamic vulnerability oracles [13]. Subsequent learning-based studies still categorise timestamp and transaction-ordering weaknesses, suggesting that these categories are still salient in benchmark construction and in practical analysis. Their persistence reflects both platform semantics and the continued use of reproducible benchmark labels.

Arithmetic/state and logic/semantic/exception weaknesses were each reported in 14 evidence sources (56%). Arithmetic categories are mainly related to overflow and underflow; logic categories are related to exception mishandling, unchecked return values, inconsistent visibility, contract-standard misuse, and state-transition errors. SmartCheck shows that pattern-based analysis of sources can identify both arithmetic and semantic defects [12]. Eth2Vec generalises the detection problem to learn code representations of multiple classes, such as arithmetic weaknesses and visibility-related issues [25]. The data indicate that these families are relatively susceptible to automated detection, but the variation in labels and benchmarks prevents the direct comparison of the reported performance.

Lifecycle / fund-safety weaknesses were reported in 12 of the 25 included evidence sources (48%), whereas authorization / access-control and resource/availability weaknesses were each reported in 11 sources (44%). Suicidal or destructible contracts, locked Ether, unlimited money flow, and unsafe value-transfer paths are all examples of lifecycle risks. MAIAN distinguishes stateful greedy, prodigal, and suicidal behaviors that cannot be fully modeled as local source-level defects [15]. The access-control weaknesses are: misuse of the tx. origin, unlimited privileged functions, lack of visibility restrictions, and weak authorisation. Risks in availability are gas exhaustion, denial of service, and infinite loops. These lower reporting frequencies should not be interpreted as lower real-world severity; they indicate how often the categories were directly examined in the selected literature. Table **4** summarises the taxonomy, and Fig. (**3**) the cross-family synthesis framework.

3.3. Detection, Prevention, Verification, and Mitigation Approaches

The included evidence sources demonstrate three broad methodological stages. The first stage focuses on symbolic execution, static analysis, fuzzing, and exploit generation. Oyente demonstrates symbolic execution over Ethereum contracts [10]. SmartCheck instantiates vulnerability knowledge as source-level patterns [12], and Securify as dependency graphs and security patterns [14]. ContractFuzzer introduces execution-based testing with vulnerability-specific oracles [13]. MAIAN, teEther, and ETHBMC also show that multi-transaction or exploitable states can be determined by symbolic and bounded reasoning as opposed to local syntax alone. Together, these methods form a robust basis for pre-deployment analysis, though their reach is limited by the semantics of the modeled EVM, path exploration, vulnerability specifications, and available computational resources.

A second stage deals with deployed or operational contracts. Instead of having to replace contracts, Sereum defends against reentrancy by runtime monitoring and dynamically tracking tainted contracts [17]. This shifts the mitigation goal from finding a defect before deployment to limiting damaging execution after deployment. A complementary empirical view is given by Perez and Livshits [22], who demonstrate that a small share of the contracts that were deemed vulnerable were actually exploited on-chain. Through their analysis, they justify the need to keep vulnerability detection, exploitability, and realised harm analytically separate. Even a technically correct

detector can produce findings with substantially different practical risk, which supports exploitability-aware prioritisation and validation.

The third stage is defined by machine and deep learning. Graph-based learning represents contract structure and control relations, as demonstrated by Zhuang *et al.* [20]. Multi-task learning shares information across vulnerability classes [23], and CBGRU employs convolutional and recurrent representations [24]. Eth2Vec trains contract-wide representations that can remain useful under code rewriting [25]. Subsequent works augment the richness of representation by hierarchical graphs, semantic pretrained models, multimodal fusion, and heterogeneous AST-CFG-DFG graphs. Ma *et al.* [28] employed hierarchical attention, Tang *et al.* [29] reported high-performance results for a CodeBERT-based approach, and Deng *et al.* [30] used a combination of multiple code modalities. Xu *et al.* [31] expand graph learning to contract-level and line-level detection, indicating a shift toward finer-grained vulnerability localisation alongside classification and explainability.

Another form of security-assurance evidence is provided by secondary syntheses and taxonomies. Kushwaha *et al.* [26] and Zhou *et al.* [27] categorise vulnerabilities, tools and countermeasures within the Ethereum ecosystem. OpenSCV addresses terminology fragmentation by creating a hierarchical vulnerability taxonomy using heterogeneous classifications and detection tools [32]. Bresil *et al.* [33] compare deep-learning strategies based on the published evidence. In contrast, Azimi *et al.* [34] review the security design patterns and discover that a limited number of them directly address the known types of vulnerabilities. Together, these sources suggest that detection research has grown faster than preventive design coverage. Table **5** therefore distinguishes methodological families by control point, evidence base, and recurring limitations rather than treating detection, prevention, verification, and mitigation methods as equivalent.

3.4. Vulnerability-to-Security-Approach Cross-Map

According to the cross-map, smart-contract security assurance is layered. Design-time checks, effects, interactions, and reentrancy guards address external-call vulnerabilities, which can be detected using fuzzing during testing, reentrancy guards before deployment, or by static or symbolic methods before deployment. Checked arithmetic and secure coding can help eliminate arithmetic weaknesses, although it is also a goal of a static analyser and learning-based classifier. Block-context weaknesses either involve code-level limitations and the reasoning of transaction or environmental state, and lifecycle and fund-safety defects can frequently involve trace or value-flow analysis. This trend implies that the general security assurance cannot be narrowed down to the option of using the most successful detector in a single benchmark. Table **6** associates the seven vulnerability families with the most defensible detection, prevention, verification, and mitigation strategies embodied in the corpus.

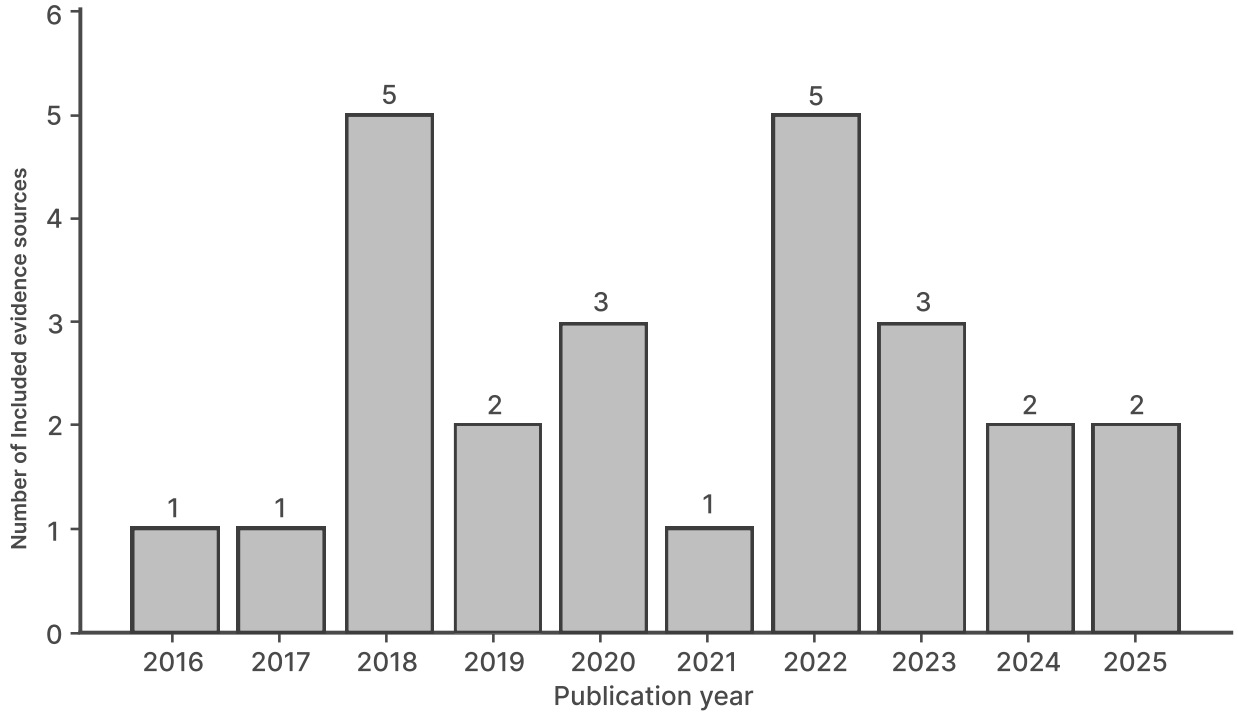


Fig. (2). Publication-year distribution of the included evidence sources.

Note: Author-developed figure based on the publication years of the 25 included evidence sources. The corpus shows two local peaks, in 2018 and 2022, followed by sustained work on learning-based and graph-based vulnerability detection through 2025.

Table 4. Document-level reporting frequency of vulnerability families.

Vulnerability Family	Operational Scope	Evidence Sources, n (%)	Interpretive Note
External-call / control-flow	Reentrancy; unsafe calls; delegatecall/callstack; unchecked low-level interactions	23 (92%)	Most persistent family across symbolic, static, dynamic, runtime, and learning-based research.
Block/transaction context	Timestamp dependence; transaction ordering; front-running; weak randomness; block-state assumptions	19 (76%)	Often depends on execution environment or transaction context rather than isolated code syntax.
Arithmetic/state	Overflow; underflow; unsafe arithmetic; state-manipulation defects	14 (56%)	Frequently represented in benchmark datasets and multi-class detectors.
Logic / semantic / exception	Unchecked returns; exception disorder; visibility/semantic errors; standard noncompliance	14 (56%)	Labels vary considerably across tools, which reduces direct comparability.
Lifecycle/fund safety	Suicidal/destructible contracts; locked Ether; unrestricted money flow; greedy/prodigal behavior	12 (48%)	Often requires trace-level or value-flow reasoning.
Authorisation/access control	tx.origin; missing authorisation; unrestricted privileged functions; visibility flaws	11 (44%)	Mitigation depends on both coding discipline and analysis of privileged operations.
Resource/availability	Gas exhaustion; denial of service; infinite loops; costly execution	11 (44%)	May depend on environmental limits and runtime behavior.

Note: *An included publication could contribute to more than one vulnerability family, so percentages do not sum to 100%. Six of the 25 evidence sources were secondary reviews, syntheses, or taxonomies. These sources were counted once at the document level and were used for taxonomy harmonisation and contextual corroboration; they were not treated as independent empirical replications of primary-study findings. Because some primary studies may also be discussed within secondary sources, the frequencies describe literature coverage rather than counts of independent empirical datasets and should not be interpreted as population prevalence, attack incidence, or comparative effect estimates.*

Table 5. Synthesis of detection, prevention, verification, and mitigation families.

Family	Typical Control Point	Evidence Represented in the Corpus	Main Strength	Recurring Limitation
Static analysis/security patterns	Pre-deployment source or bytecode	Pattern rules, IR/data-flow, dependency graphs	Fast, automatable screening across large corpora	Rule coverage and false positives depend on specification quality.
Symbolic execution / formal methods	Pre-deployment and exploit validation	Path exploration, bounded model checking, trace properties	Can reason about execution states and prove/validate specific properties	Path explosion, modelling cost, and incomplete semantic coverage.
Fuzzing / dynamic testing	Pre-deployment execution	Input generation and vulnerability-specific oracles	Observes runtime behavior and can reveal execution-dependent defects	Coverage depends on generated inputs, instrumentation, and oracle design.
Runtime monitoring	Post-deployment execution	Dynamic reentrancy protection and execution monitoring	Protects already deployed contracts without source changes	May target narrow vulnerability classes and add runtime complexity.
Machine / deep learning	Primarily pre-deployment detection	Graph, sequence, embedding, pretrained-code and multimodal models	Scales multi-class detection and learns complex representations	Benchmark dependence, label quality, distribution shift, and explainability.
Secure coding/design patterns	Design and implementation	Preventive coding practices and security patterns	Reduces vulnerability introduction before analysis is needed	Direct pattern coverage remains limited relative to the vulnerability landscape.
Auditing/benchmarking/taxonomy	Cross-cutting assurance	Tool comparison, exploitation analysis, normalised taxonomies	Improves interpretation, prioritisation, and comparability	Heterogeneous tool outputs and taxonomies remain difficult to normalise.

Table 6. Cross-mapping of vulnerability families to detection, prevention, verification, and mitigation approaches.

Vulnerability Family	Supported Controls / Methods	Representative Evidence	Interpretation
External-call / control-flow	Checks-effects-interactions; reentrancy guards; static/symbolic analysis; fuzzing; runtime monitoring; graph/ML detection	Luu *et al.* [10]; Jiang *et al.* [13]; Rodler *et al.* [17]; Xu *et al.* [31]	Strongest multi-layer coverage, but persistent recurrence shows no single control is sufficient.
Block/transaction context	Avoid unsafe timestamp/randomness assumptions; transaction-aware analysis; fuzzing; graph/ML detection	Atzei *et al.* [11]; Zhuang *et al.* [20]; Bresil *et al.* [33]	Controls must model execution context and ordering, not source syntax alone.
Arithmetic / state	Checked arithmetic; static rules; formal reasoning; multi-class ML/DL	Tikhomirov *et al.* [12]; Huang *et al.* [23]; Ma *et al.* [28]	Well-represented in benchmark datasets, but compiler and language changes affect relevance.
Logic / semantic / exception	Return-value checks; explicit error handling; semantic/static analysis; fuzzing; learning-based classification	Jiang *et al.* [13]; Tsankov *et al.* [14]; Tang *et al.* [29]	Taxonomy inconsistency is a major barrier to comparing tools.
Lifecycle/fund safety	Restrict destructive/value-transfer operations; trace analysis; exploit generation; model checking.	Nikolic *et al.* [15]; Krupp & Rossow [16]; Frank *et al.* [19]	Trace-level reasoning is often required because risk emerges across transactions or value flow.
Authorisation / access control	Explicit authorisation; visibility control; role-based patterns; static/ML analysis	Huang *et al.* [23]; Xu *et al.* [31]; Azimi *et al.* [34]	Prevention is design-sensitive and benefits from privileged-operation analysis.
Resource/availability	Bound loops; gas-aware design; static/dynamic analysis; learning-based detection	Tikhomirov *et al.* [12]; Zhuang *et al.* [20]; Xu *et al.* [31]	Availability risk depends on platform limits and runtime resource behavior.

4. DISCUSSION

4.1. Persistent Vulnerability Structure

The synthesis provides the answer to RQ1 by demonstrating that the research on smart-contract security is highly focused on external calls, control flow, and transaction context. At the document level, external-call/control-flow weaknesses were reported in 92% of the included evidence sources, and block/transaction-context weaknesses were reported in 76%. The prominence of these families is technically consistent since smart contracts are a mixture of program logic and externally triggered calls, publicly ordered transactions, economic transfer of value, and deterministic execution on a shared state. Atzei *et al.* [11] already conceptualised many attacks as language, EVM, and blockchain layer interactions, and subsequent evidence does not refute that framework. Rather, more modern approaches recode the same underlying flaws in various forms again and again. This continuity suggests that improvements have been made in analysis capability, scale, and representation, as opposed to the removal of the underlying classes of smart-contract risk.

The frequency analysis also explains why arithmetic or access-control bugs cannot be considered obsolete merely because more recent languages or compiler checkers address some instances. At the document level, arithmetic/state weaknesses were reported in 56% of the included evidence sources, and authorisation/access-control weaknesses were reported in 44%. This continuity is partially due to benchmark continuity, but also demonstrates that security failures are not entirely compiler problems. Huang *et al.* [23] utilise arithmetic and reentrancy as targets of multi-task detection, and MVD-HG consists of arithmetic and access-control targets in a more general heterogeneous-graph framework [31]. The field thus still considers these classes of vulnerabilities as useful analytical categories regardless of the change in the implementation environment.

These frequencies require careful interpretation because six included sources are secondary reviews, syntheses, or taxonomies that may discuss primary studies also represented directly in the corpus. Secondary sources were retained to stabilise terminology, corroborate vulnerability categories, and map broader methodological coverage, but they were not treated as additional independent empirical confirmations. Repeated reporting of a vulnerability family in a primary study and in a secondary synthesis therefore indicates convergence in the literature rather than empirical replication. The cross-map should consequently be interpreted as a descriptive synthesis of recurring vulnerability-method relationships, with primary studies providing the principal empirical basis and secondary sources providing contextual corroboration and taxonomy harmonisation.

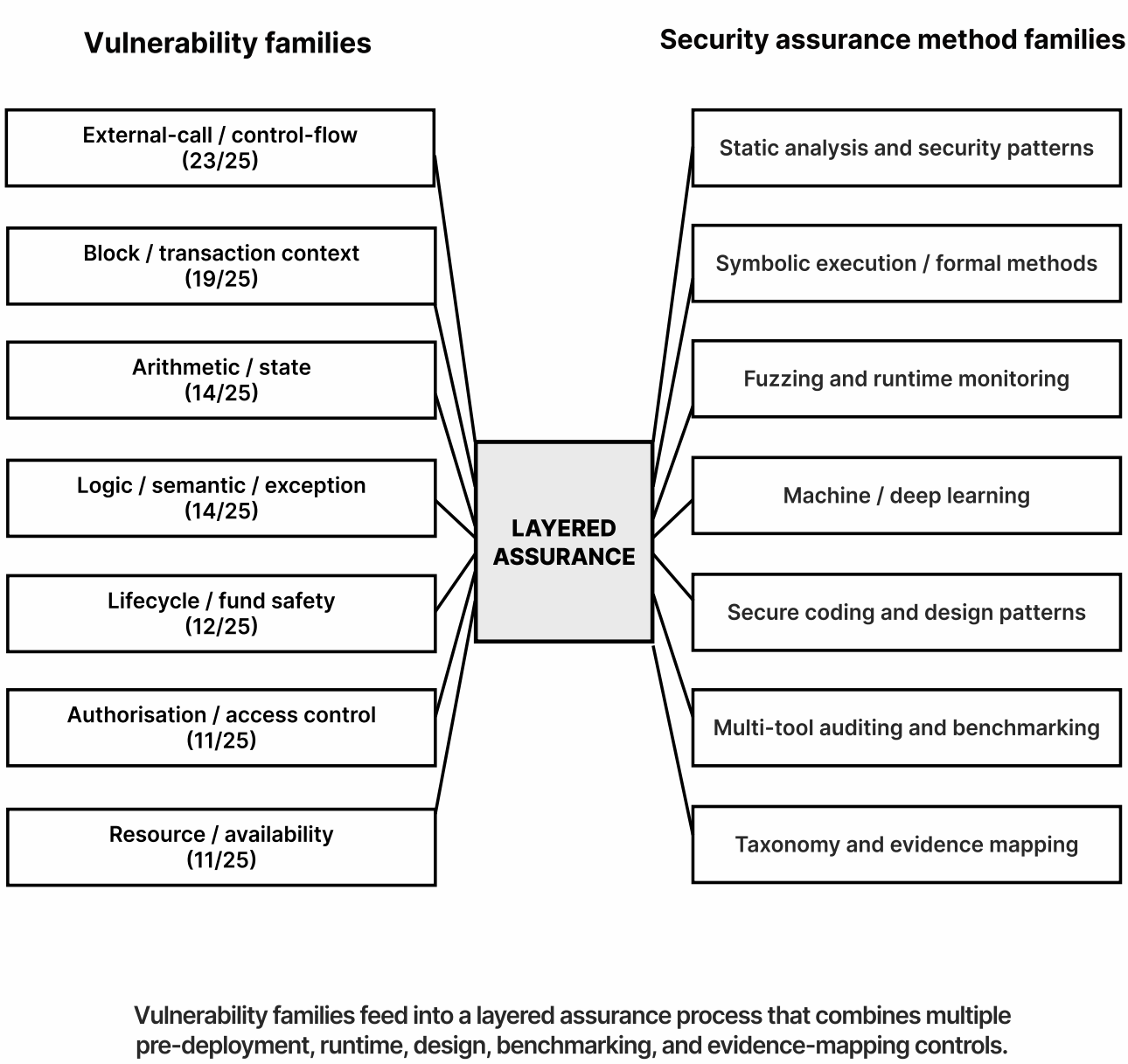


Fig. (3). Evidence-based vulnerability and security-assurance synthesis framework.

Note: Author-developed figure based on evidence-source-level coding. Security-assurance method families encompass detection, prevention, verification, and mitigation approaches. Arrows indicate that each vulnerability family may require multiple complementary control families; they do not imply exclusive one-to-one relationships. Counts in the framework represent document-level coverage and do not indicate independent empirical replication or effect magnitude.

4.2. Evolution from Rule-Based Analysis to Learned Representations

The obvious methodological shift over the decade provides an answer to RQ3. Initial studies focus on symbolic execution, pattern-based static analysis, fuzzing, dependency-graph analysis, and exploit generation. These techniques expose their assumptions and tend to leave traceable results or rule violations. An example of symbolic execution is provided by Luu *et al.* [10], and formalization of source-level patterns is implemented in SmartCheck [12]. teEther generates concrete exploits, whereas ETHBMC identifies bounded counterexamples or triggering inputs [16, 19]. Interpretability is the key advantage of this generation, and scalability and semantic completeness are recurring limitations.

Representation learning has gained more and more prominence since 2020. Graph neural networks are able to

capture structural relationships, multi-task models are able to share information across vulnerability classes, and code-language models learn semantic representations. Tang *et al.* [29] and Deng *et al.* [30] indicate that an optimized CodeBERT-based model has good F1 performance, and that integrating source, opcode, and control-flow information can enhance multi-class detection, respectively. MVD-HG continues this orientation with heterogeneous graph relations as well as line-level localisation [31]. Nevertheless, improved benchmark performance does not always result in stronger operational security. Results based on learning are sensitive to labels, datasets, injection strategies, compiler versions, and train-test distributions; the values of performance must be viewed as evidence conditioned on the benchmark but not as evidence of universal superiority.

4.3. Effectiveness, Validation, and Exploitability

RQ2 requires distinguishing detection, prevention, verification, and mitigation from demonstrated risk reduction. Durieux *et al.* [21] demonstrate that even within the same population of contracts, automated analysers can diverge significantly, which implies that the output of tools is sensitive to implementation, supported compiler versions, as well as vulnerability definitions. Perez and Livshits [22] introduce another critical risk factor: only a small proportion of the contracts identified as vulnerable in their large sample were observed to be exploited (1.98%). This does not diminish the importance of vulnerability detection; rather, it shows that exposure, exploitability, attacker incentives, and contract value mediate realised harm. A stronger assurance process should therefore integrate vulnerability identification with exploit validation, contextual severity assessment, and mitigation verification.

Design- and runtime-oriented evidence also demonstrates the necessity of layered controls. Sereum is able to defend against reentrancy in existing contracts during execution time [17], but design patterns aim to defend against unsafe structures prior to deployment. But Azimi *et al.* [34] conclude that only a small number of established security patterns can be directly related to recognised vulnerability categories, which suggests a prevention-coverage gap. OpenSCV also illustrates the scope and terminological disaggregation of the vulnerability domain [32]. These results suggest that security engineering cannot be based on one lifecycle stage. Prevention, detection, formal reasoning, exploit validation, and runtime safeguards deal with various failure modes and should be integrated based on the criticality of the contract.

4.4. Research Gaps and Implications

There are four significant evidence gaps that are identified in RQ4. First, the corpus is largely Ethereum- and Solidity-focused, which means that the generalizability across platforms is low. Second, the comparability of benchmarking is limited due to the fact that the evidence sources contain various vulnerability labels, injected defects, real-world contracts, compiler versions, and metrics. Third, numerous learning-based primary studies maximize classification performance but do not have similar evidence on exploitability, remediation efficacy, or post-deployment behavior. Fourth, the coverage of preventive design is still significantly less than the list of known vulnerabilities. Future reports should give standardised definitions of vulnerabilities, provenance of data sets, compiler and platform versions, class distribution, false-positive and false-negative analysis, and reproducible code or benchmark artifacts where feasible.

The results also carry practical implications for auditors and developers. Coverage can be fast with static analysis, but it should be complemented with analyses of vulnerabilities that are sensitive to multi-transaction state or runtime circumstances. Symbolic and formal methods provide more powerful execution reasoning, but can have more limited specifications and demand more computation. Fuzzing provides tangible evidence of execution, and runtime protection can decrease the risk of already deployed contracts. Learning-based models are also scale- and multi-class-sensitive, though when the contract value is high, they should be used in combination with rule-based or semantic checks. The cross-map in Table **6** therefore supports a defense-in-depth strategy in which techniques should be selected according to the vulnerability family, lifecycle phase, and required assurance level.

LIMITATIONS AND FUTURE DIRECTIONS

This review has several limitations. Only four information sources and English-language literature were searched, which means that potentially eligible work that is indexed elsewhere or published in other languages might not have been found. The title-based search query limited the amount of general blockchain-security literature, but this method can suppress sensitivity to articles whose titles focus on a tool name or platform-specific terms instead of security or vulnerability terms. Titles and abstracts were screened and full texts evaluated by two reviewers who discussed and agreed on disagreements, though no formal inter-rater reliability coefficient was computed. The numerical study-selection pathway and full-text exclusion categories applied in this review are reported in Fig. (**1**).

The final corpus contained 25 evidence sources from 1,200 identified records, comprising 19 primary technical or empirical studies and six secondary reviews, syntheses, or taxonomies. This substantial reduction reflects the strict eligibility criteria requiring direct smart-contract security relevance, peer review, readable full text, sufficient extractable technical detail, English language, and publication within 2016–2025. These restrictions increased topical specificity but reduced empirical breadth and may have omitted relevant work with tool-specific titles, incomplete reporting, non-English publication, or indexing outside the four searched sources. Accordingly, the vulnerability-to-security-method cross-map should be interpreted as a structured synthesis of the eligible evidence captured by this protocol rather than a comprehensive or universally generalisable representation of all smart-contract security research. The limited primary-study base also reduces confidence in sparsely represented vulnerability-method associations and cross-platform conclusions.

The evidence included is heterogeneous, and primary software tools, benchmark studies, empirical on-chain analyses, taxonomies, systematic reviews, and meta-analytic comparisons do not offer directly equivalent types of evidence. No formal risk-of-bias score was used since no single appraisal tool is suitable for all these designs; evidence type and validation basis were charted directly. Document-level frequencies indicate how often a family of vulnerabilities is reported in the sources of evidence included in the analysis and cannot be understood as actual prevalence of vulnerabilities or attack rates. Performance measures should also not be compared across incomparable datasets as though they had been measured in equal conditions. Lastly, the research on Ethereum, Solidity, and EVM is predominant, which restricts the inferences to other smart-contract ecosystems. These restrictions do not obstruct evidence mapping, but limit conclusions regarding universal security efficacy. Although secondary sources were not given independent empirical weight, overlap remains possible because some of their underlying primary studies may also appear separately in the corpus. This further support interpreting the frequencies and cross-map as descriptive evidence mapping rather than quantitative estimates of independent evidence strength. Future reviews could broaden database coverage, relax title-field constraints, or conduct primary-study-only synthesis where sufficiently comparable data permit.

CONCLUSION

This scoping review maps smart-contract security evidence published from 2016 to 2025 and identifies a persistent vulnerability structure despite rapid methodological change. External-call/control-flow weaknesses were the most frequently reported family, followed by block/transaction-context weaknesses, with arithmetic, semantic, lifecycle, authorisation and availability risks being recurrent throughout the decade. Graph-based, sequence-based, pretrained-code and multimodal learning methods replaced explicit symbolic, static, fuzzing, and runtime methods as they became increasingly integrated into the literature. However, benchmark performance does not offer practical risk mitigation in itself, and empirical evidence of exploitation shows that vulnerability detection ought not necessarily be taken to be evidence of realised exploitation. Within the evidence captured by this review, the cross-map supports a layered assurance approach incorporating safe design, automated analysis, formal or symbolic reasoning where appropriate, dynamic validation, and runtime protection. Because the corpus is limited and combines primary with secondary evidence, the mapped associations should be interpreted as recurring patterns in the eligible literature rather than universal estimates of security-method effectiveness. Future research should broaden database and platform coverage where feasible and consider primary-study-only synthesis when sufficiently comparable data are available, alongside standardised benchmarks, cross-platform testing, exploitability-aware evaluation, reproducible testing, and preventive mechanisms that span a larger fraction of the known vulnerability space.

LIST OF ABBREVIATIONS

AST	=	Abstract Syntax Tree
CFG	=	Control-Flow Graph
DL	=	Deep Learning
DFG	=	Data-Flow Graph
EVM	=	Ethereum Virtual Machine
ETHBMC	=	Ethereum Bounded Model Checker
ML	=	Machine Learning
PRISMA-ScR	=	Preferred Reporting Items for Systematic Reviews and Meta-Analyses Extension for Scoping Reviews
MVD-HG	=	Multigranularity Vulnerability Detection based on Heterogeneous Graphs

AUTHOR'S CONTRIBUTION

A.M.G. was responsible for conceptualization, methodology, investigation, data curation, formal analysis, writing-original draft preparation, and writing-review and editing. The author read and approved the final version of the manuscript.

ETHICAL APPROVAL AND INFORMED CONSENT

Ethical approval was not required because this scoping review synthesized published literature and involved no primary data collection from human participants or animals.

REPORTING OF STANDARDS

PRISMA guidelines have been followed for this study.

AVAILABILITY OF DATA AND MATERIALS

All data synthesised in this review are derived from the published evidence sources cited in the manuscript; the extracted study characteristics and synthesis outputs are presented in the tables and appendices.

FUNDING

None.

CONFLICT OF INTEREST

The author declare that they have no conflict of interests.

ACKNOWLEDGEMENTS

Declared none.

DECLARATION OF AI

Author only used an AI assistant (Claude, Anthropic) as an aid to organizing the literature, drafting, and writing of the prototype code. The author thoroughly guided and revised this whole procedure. Author completely accept culpability of the accuracy, originality, and integrity of the manuscript and whatever it comprises.

APPENDICES

Appendix A

Scopus Search Evidence

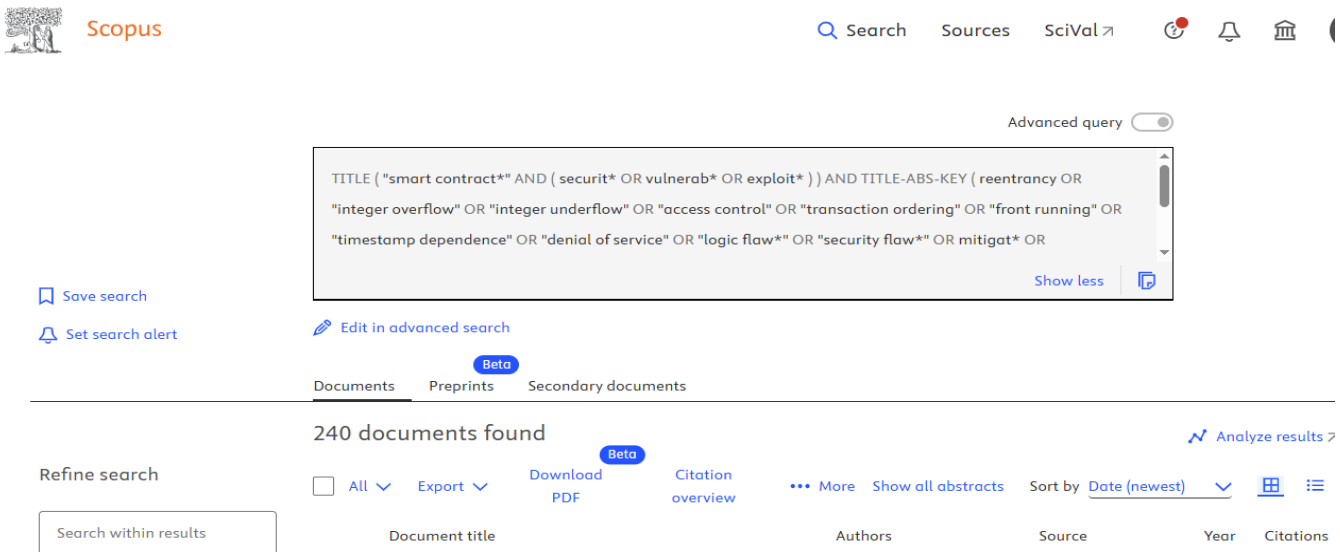


Fig. (A1). Final Scopus search interface showing 240 retrieved documents.

Appendix B

Web of Science Search Evidence

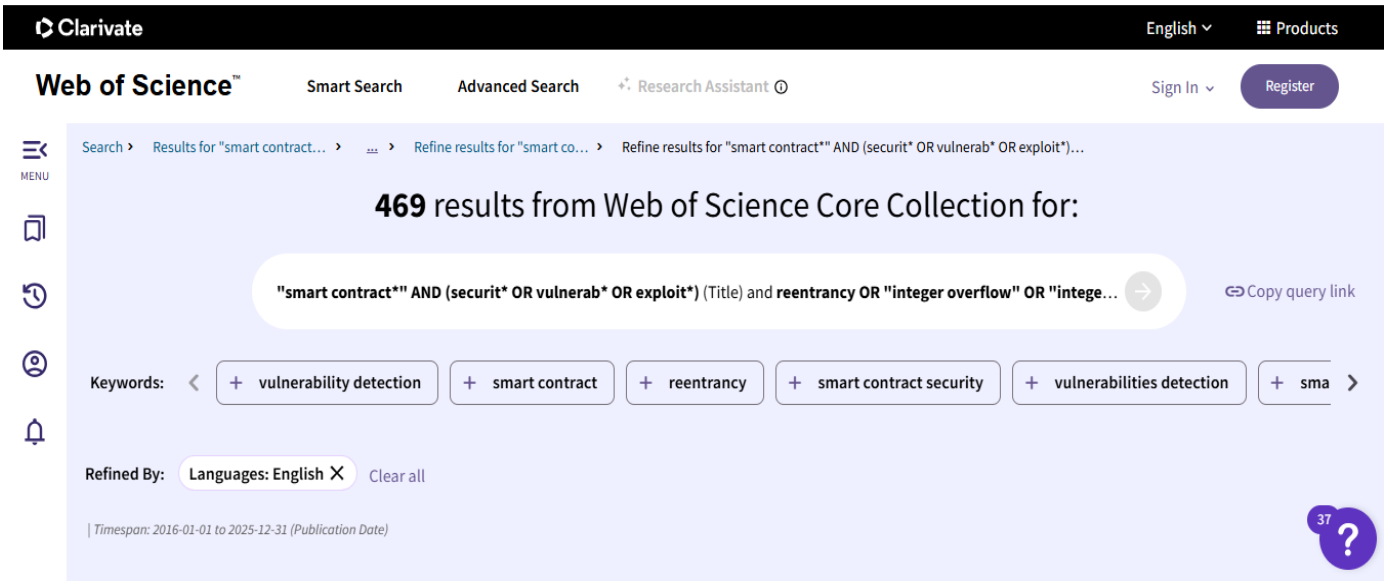


Fig. (B1). Final Web of Science Core Collection result page showing 469 records, English language, and the complete 2016-01-01 to 2025-12-31 period.

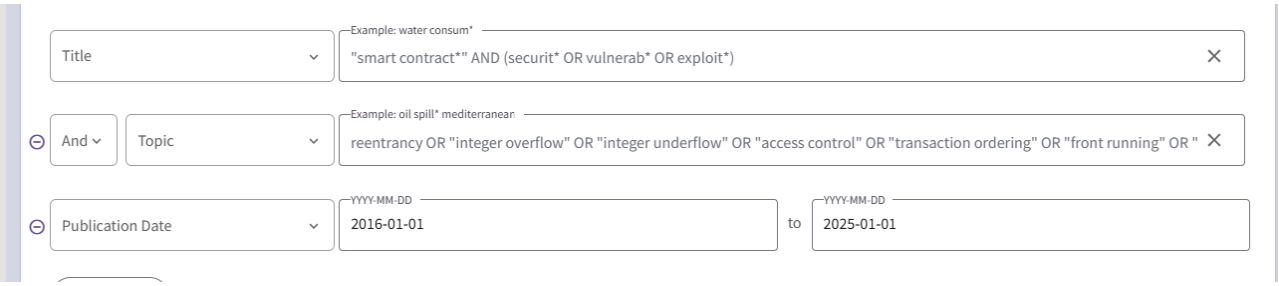


Fig. (B2). Web of Science fielded-search interface used to separate the Title and Topic concept blocks.

Appendix C

IEEE Xplore Search Evidence

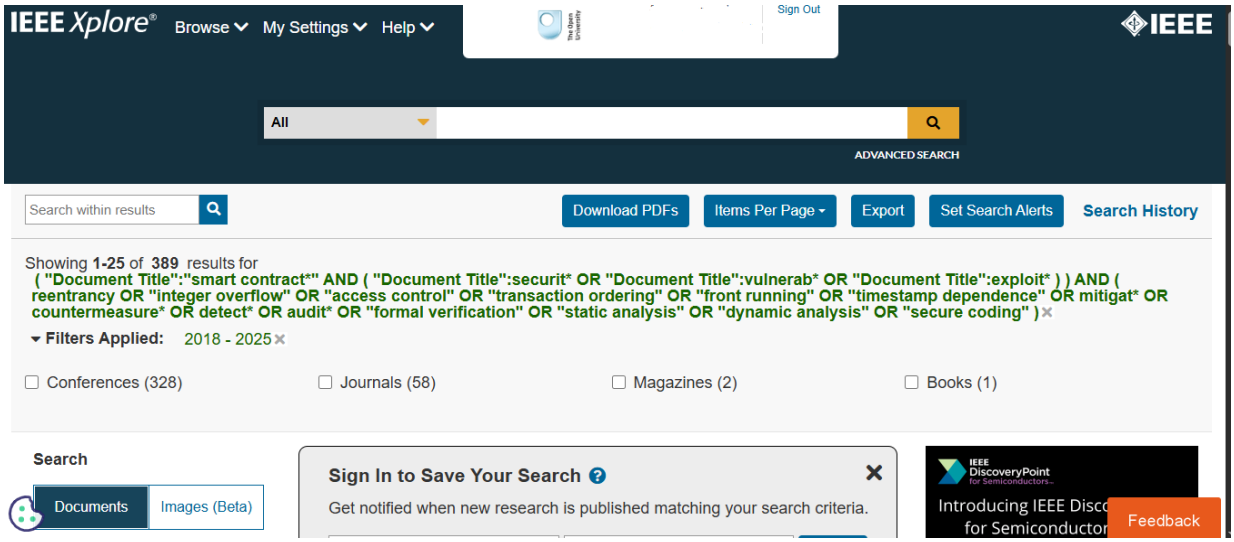


Fig. (C1). Final IEEE Xplore results page showing 389 records and the interface-displayed 2018–2025 result range.

Appendix D

Springer Nature Link Search Evidence

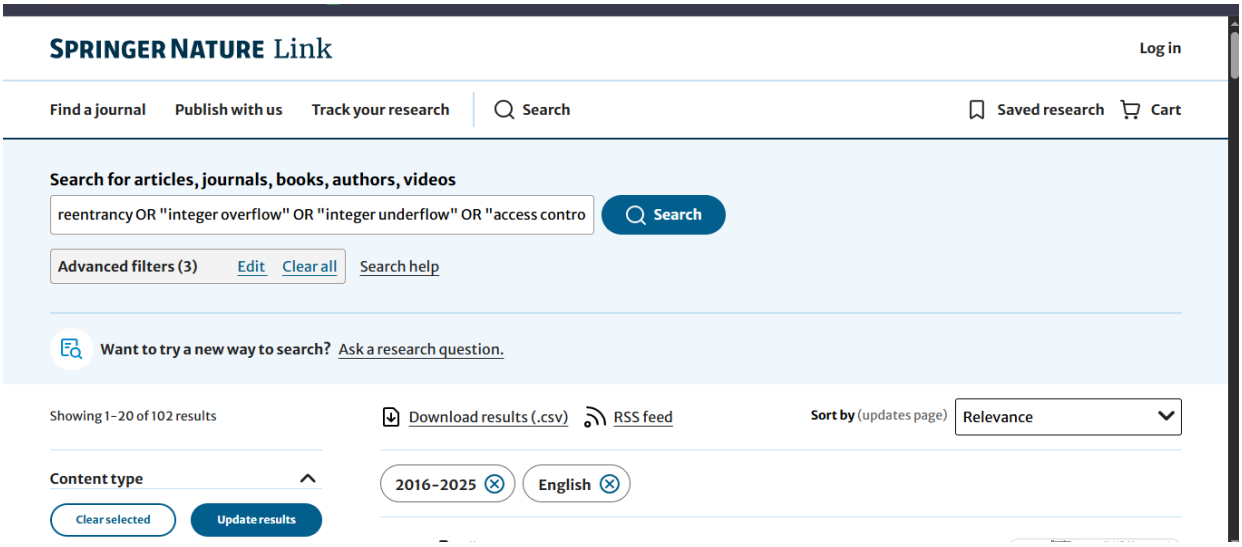


Fig. (D1). Final Springer Nature Link result page showing 102 results with 2016–2025 and English filters.

Appendix E

PRISMA-ScR Reporting Crosswalk

This crosswalk maps the principal PRISMA-ScR reporting domains to their locations in the manuscript and identifies how each domain is reported.

Reporting Domain	Location	Reporting Status / Note
Title identifies review type	Title	The title explicitly identifies the work as a scoping review.
Abstract	Abstract	Objectives, information sources, period, corpus size, synthesis approach, principal findings, and conclusion are summarised.
Rationale	Sections 1.1–1.2	Security context and the evidence-synthesis gap are defined.
Objectives and questions	Sections 1.3–1.4	Purpose, contributions, and four research questions are specified.
Protocol and registration	Section 2.1	No prospective protocol was registered; this is stated explicitly.
Eligibility criteria	Section 2.2; Table **1**	Publication period, language, publication status, technical focus, evidence type, full-text availability, and version control are operationalised.
Information sources	Section 2.3; Table **2**	Four sources, search date, fields, periods, language handling, and identification counts are reported.
Electronic search strategy	Section 2.4; Appendices **A**–**D**	Database-specific search syntax and interface evidence are provided.
Selection process	Section 2.5; Fig **1**	Duplicate removal, title/abstract screening, retrieval, full-text assessment, exclusion reasons, and final inclusion are numerically reconciled.
Data charting	Section 2.6; Table **3**	Bibliographic, technical, validation, benchmark, and contribution variables are charted.
Coding and synthesis	Section 2.7and Tables **4**–**6**	Study-level coding assignments are traceable.
Evidence appraisal approach	Section 2.8	Formal risk-of-bias scoring is not used because designs are heterogeneous; validation basis and evidence type are charted instead.
Selection results	Section 3.1; Fig. **1**	The 1,200-record identification pool and 25-source final corpus are reported with intermediate counts.
Characteristics of evidence	Table **3**	All 25 included sources are summarised by design, platform, vulnerability focus, method, validation, and contribution.
Synthesis results	Sections 3.2–3.4; Tables **4**–**6**; Figs. **2**–**3**	Document-level frequency, security-assurance method families, temporal patterns, and vulnerability-to-security-method cross-mapping are reported.
Discussion	Section 4	Findings are interpreted against the research questions, methodological evolution, validation, exploitability, and practical assurance.
Limitations	Section 5	Search sensitivity, language/source restrictions, screening reliability, heterogeneous designs, and platform concentration are addressed.
Conclusions	Section 6	The conclusion synthesizes the principal findings and future research priorities without duplicating earlier sections.

REFERENCES

[1] H. Chen, M. Pendleton, L. Njilla, and S. Xu, "A survey on Ethereum systems security: Vulnerabilities, attacks, and defenses," ACM Comput. Surv., vol. 53, no. 3, Art. no. 67, pp. 1–43, 2020, https://doi.org/10.1145/3391195

[2] A. Singh, R. M. Parizi, Q. Zhang, K.-K. R. Choo, and A. Dehghantanha, "Blockchain smart contracts formalization: Approaches and challenges to address vulnerabilities," Comput. Secur., vol. 88, Art. no. 101654, 2020, https://doi.org/10.1016/j.cose.2019.101654

[3] P. Tolmach, Y. Li, S.-W. Lin, Y. Liu, and Z. Li, "A survey of smart contract formal specification and verification," ACM Comput. Surv., vol. 54, no. 7, pp. 1–38, 2021, https://doi.org/10.1145/3464421

[4] N. Kannengießer, S. Lins, C. Sander, K. Winter, H. Frey, and A. Sunyaev, "Challenges and common solutions in smart contract development," IEEE Trans. Softw. Eng., vol. 48, no. 11, pp. 4291–4318, 2022, https://doi.org/10.1109/TSE.2021.3116808

[5] H. Chu, P. Zhang, H. Dong, Y. Xiao, S. Ji, and W. Li, "A survey on smart contract vulnerabilities: Data sources, detection and repair," Inf. Softw. Technol., vol. 159, Art. no. 107221, 2023, https://doi.org/10.1016/j.infsof.2023.107221

[6] H. Arksey and L. O'Malley, "Scoping studies: Towards a methodological framework," Int. J. Soc. Res. Methodol., vol. 8, no. 1, pp. 19–32, 2005, https://doi.org/10.1080/1364557032000119616

[7] M. D. J. Peters *et al.,* "Updated methodological guidance for the conduct of scoping reviews," JBI Evid. Synth., vol. 18, no. 10, pp. 2119–2126, 2020, https://doi.org/10.11124/JBIES-20-00167

[8] A. C. Tricco *et al.,* "PRISMA extension for scoping reviews (PRISMA-ScR): Checklist and explanation," Ann. Intern. Med., vol. 169, no. 7, pp. 467–473, 2018, https://doi.org/10.7326/M18-0850

[9] Z. Munn, M. D. J. Peters, C. Stern, C. Tufanaru, A. McArthur, and E. Aromataris, "Systematic review or scoping review? Guidance for authors when choosing between a systematic or scoping review approach," BMC Med. Res. Methodol., vol. 18, Art. no. 143, 2018, https://doi.org/10.1186/s12874-018-0611-x

[10] L. Luu, D.-H. Chu, H. Olickel, P. Saxena, and A. Hobor, "Making smart contracts smarter," in Proc. 2016 ACM SIGSAC Conf. Comput. Commun. Secur. (CCS), 2016, pp. 254–269, https://doi.org/10.1145/2976749.2978309

[11] N. Atzei, M. Bartoletti, and T. Cimoli, "A survey of attacks on Ethereum smart contracts (SoK)," in Principles of Security and Trust, M. Maffei and M. Ryan, Eds. Cham, Switzerland: Springer, pp. 164–186, 2017, https://doi.org/10.1007/978-3-662-54455-6_8

[12] S. Tikhomirov, E. Voskresenskaya, I. Ivanitskiy, R. Takhaviev, E. Marchenko, and Y. Alexandrov, "SmartCheck: Static analysis of Ethereum smart contracts," in Proc. 1st Int. Workshop Emerg. Trends Softw. Eng. Blockchain (WETSEB), 2018, pp. 9–16, https://doi.org/10.1145/3194113.3194115

[13] B. Jiang, Y. Liu, and W. K. Chan, "ContractFuzzer: Fuzzing smart contracts for vulnerability detection," in Proc. 33rd ACM/IEEE Int. Conf. Autom. Softw. Eng. (ASE), 2018, pp. 259–269, https://doi.org/10.1145/3238147.3238177

[14] P. Tsankov, A. Dan, D. Drachsler-Cohen, A. Gervais, F. Bünzli, and M. Vechev, "Securify: Practical security analysis of smart contracts," in Proc. 2018 ACM SIGSAC Conf. Comput. Commun. Secur. (CCS), 2018, pp. 67–82, https://doi.org/10.1145/3243734.3243780

[15] I. Nikolić, A. Kolluri, I. Sergey, P. Saxena, and A. Hobor, "Finding the greedy, prodigal, and suicidal contracts at scale," in Proc. 34th Annu. Comput. Secur. Appl. Conf. (ACSAC), 2018, pp. 653–663, https://doi.org/10.1145/3274694.3274743

[16] J. Krupp and C. Rossow, "teEther: Gnawing at Ethereum to automatically exploit smart contracts," in *27th USENIX Secur. Symp. (USENIX Security 18)*, 2018, pp. 1317–1333, Available from: https://www.usenix.org/conference/usenixsecurity18/presentation/krupp

[17] M. Rodler, W. Li, G. O. Karame, and L. Davi, "Sereum: Protecting existing smart contracts against re-entrancy attacks," in Proc. Netw. Distrib. Syst. Secur. Symp. (NDSS), 2019, https://doi.org/10.14722/ndss.2019.23413

[18] J. Feist, G. Grieco, and A. Groce, "Slither: A static analysis framework for smart contracts," in Proc. IEEE/ACM 2nd Int. Workshop Emerg. Trends Softw. Eng. Blockchain (WETSEB), 2019, pp. 8–15, https://doi.org/10.1109/WETSEB.2019.00008

[19] J. Frank, C. Aschermann, and T. Holz, "ETHBMC: A bounded model checker for smart contracts," in *29th USENIX Secur. Symp. (USENIX Security 20)*, pp. 2757–2774, 2020, Available from: https://www.usenix.org/conference/usenixsecurity20/presentation/frank

[20] Y. Zhuang, Z. Liu, P. Qian, Q. Liu, X. Wang, and Q. He, "Smart contract vulnerability detection using graph neural network," in Proc. 29th Int. Joint Conf. Artif. Intell. (IJCAI), 2020, pp. 3283–3290, https://doi.org/10.24963/ijcai.2020/454

[21] T. Durieux, J. F. Ferreira, R. Abreu, and P. Cruz, "Empirical review of automated analysis tools on 47,587 Ethereum smart contracts," in Proc. ACM/IEEE 42nd Int. Conf. Softw. Eng. (ICSE), 2020, pp. 530–541, https://doi.org/10.1145/3377811.3380364

[22] D. Perez and B. Livshits, "Smart contract vulnerabilities: Vulnerable does not imply exploited," in *30th USENIX Secur. Symp. (USENIX Security 21)*, pp. 1325–1341, 2021, Available from: https://www.usenix.org/conference/usenixsecurity21/presentation/perez

[23] J. Huang, K. Zhou, A. Xiong, and D. Li, "Smart contract vulnerability detection model based on multi-task learning," Sensors, vol. 22, no. 5, Art. no. 1829, 2022, https://doi.org/10.3390/s22051829

[24] L. Zhang *et al.,* "CBGRU: A detection method of smart contract vulnerability based on a hybrid model," Sensors, vol. 22, no. 9, Art. no. 3577, 2022, https://doi.org/10.3390/s22093577

[25] N. Ashizawa, N. Yanai, J. P. Cruz, and S. Okamura, "Eth2Vec: Learning contract-wide code representations for vulnerability detection on Ethereum smart contracts," Blockchain Res. Appl., vol. 3, no. 4, Art. no. 100101, 2022, https://doi.org/10.1016/j.bcra.2022.100101

[26] S. S. Kushwaha, S. Joshi, D. Singh, M. Kaur, and H.-N. Lee, "Systematic review of security vulnerabilities in Ethereum blockchain smart contract," IEEE Access, vol. 10, pp. 6605–6621, 2022, https://doi.org/10.1109/ACCESS.2021.3140091

[27] H. Zhou, A. Milani Fard, and A. Makanju, "The state of Ethereum smart contracts security: Vulnerabilities, countermeasures, and tool support," J. Cybersecur. Priv., vol. 2, no. 2, pp. 358–378, 2022, https://doi.org/10.3390/jcp2020019

[28] C. Ma, S. Liu, and G. Xu, "HGAT: Smart contract vulnerability detection method based on hierarchical graph attention network," J. Cloud Comput., vol. 12, Art. no. 93, 2023, https://doi.org/10.1186/s13677-023-00459-x

[29] X. Tang, Y. Du, A. Lai, Z. Zhang, and L. Shi, "Deep learning-based solution for smart contract vulnerabilities detection," Sci. Rep., vol. 13, Art. no. 20106, 2023, https://doi.org/10.1038/s41598-023-47219-0

[30] W. Deng, H. Wei, T. Huang, C. Cao, Y. Peng, and X. Hu, "Smart contract vulnerability detection based on deep learning and multimodal decision fusion," Sensors, vol. 23, no. 16, Art. no. 7246, 2023, https://doi.org/10.3390/s23167246

[31] J. Xu, T. Wang, M. Lv, T. Chen, T. Zhu, and B. Ji, "MVD-HG: Multigranularity smart contract vulnerability detection method based on heterogeneous graphs," Cybersecurity, vol. 7, Art. no. 55, 2024, https://doi.org/10.1186/s42400-024-00245-5

[32] F. R. Vidal, N. Ivaki, and N. Laranjeiro, "OpenSCV: An open hierarchical taxonomy for smart contract vulnerabilities," Empir. Softw. Eng., vol. 29, Art. no. 101, 2024, https://doi.org/10.1007/s10664-024-10446-8

[33] M. Bresil, P. W. C. Prasad, M. S. Sayeed, and U. A. Bukar, "Deep learning-based vulnerability detection solutions in smart contracts: A comparative and meta-analysis of existing approaches," IEEE Access, vol. 13, pp. 28894-28919, 2025, https://doi.org/10.1109/ACCESS.2025.3532326

[34] S. Azimi, A. Golzari, N. Ivaki, and N. Laranjeiro, "A systematic review on smart contract security design patterns," Empir. Softw. Eng., vol. 30, Art. no. 95, 2025, https://doi.org/10.1007/s10664-025-10646-w