

DeepIPA: Multi-Agent Static Security Dissection of iOS IPA Binaries with MCP-Orchestrated Analysis and Evidence-Based Vulnerability Verification

Muhammad Ahmad Ali Qureshi¹, Memoona Sadaf^{2*}

¹Air University, Islamabad, Pakistan

²School of Information Technology, Murdoch University, Australia

*Corresponding Author: memoonasadaf29@gmail.com

HIGHLIGHTS

- DeepIPA integrates deterministic IPA static analysis, program/security graphs, MCP-orchestrated agents, and evidence contracts in one verification pipeline.
- A Planner–Investigator–Validator separation of duties prevents language-model confidence from serving as vulnerability proof.
- Across 25 evaluated IPA files, DeepIPA achieved precision of 1.00, recall of 0.72, F1 score of 0.84, and 80% app-level classification accuracy.
- DeepIPA recorded zero app-level false positives under the supplied evaluation threshold and achieved a 46-point clean-to-vulnerable risk separation.
- On DVIA-v2, DeepIPA reported 47 findings compared with 12 for MobSF and 0–3 for the remaining specialist comparators in the supplied benchmark.

ABSTRACT

iOS applications are distributed as IPA archives that contain Mach-O executables, frameworks, resources, property lists, entitlements, provisioning information, and third-party components. These artifacts expose substantial security-relevant information without requiring application execution, yet conventional scanners typically produce fragmented findings, and large language models can overstate vulnerabilities when reasoning is not tied to machine-checkable evidence. This paper presents DeepIPA, a multi-agent framework for static security dissection of iOS IPA binaries. The framework features

deterministic multi-engine static analysis, a normalised program and security evidence graph, tool access based on the Model Context Protocol (MCP), and a three-role workflow called Planner, Investigator, and Validator. A deterministic evidence-contract gate is a type of gate that affects outcomes, where it is known that an outcome cannot become confirmed if it is not submitted to the gate. A deterministic evidence-contract gate is a type of gate that impacts outcomes; that is, if an outcome is not submitted to the gate, that outcome will not be confirmed. In evaluation, DeepIPA was compared with the

other tools MobSF, IPA Auditor, Pavise, YARA, Gitleaks, and Semgrep on 25 IPA files. Under the specified score threshold, DeepIPA's precision was 1.00, recall was 0.72, F1 was 0.84, accuracy was 80%, and false-positive rate was 0 at the application level. It also achieved the best C-V score separation and the highest level of finding

coverage on the DVIA-v2 benchmark. The findings agree with evidence-gated multi-agent orchestration as a promising strategy for conducting static mobile security analysis with high precision, and with the unknown as a safe conclusion for evidence that needs to be confirmed during runtime.

Keywords: iOS Security, IPA, Multi-Agent LLM, Model Context Protocol, Static Analysis, Evidence Contracts

1. Introduction

Mobile applications are increasingly a primary means of authentication, payments, personal comms, health info, and enterprise access. iOS applications are typically packaged for analysis in an IPA archive that includes the application bundle, Mach-O executable, embedded frameworks, property lists, entitlements, provisioning material, resources, and configuration data on Apple platforms. Many properties are thus directly observable from the delivered artifact and are relevant to security. From static inspections, clear-text endpoints, insecure App Transport Security (ATS) configurations, embedded secrets, weak cryptographic indicators, excessive entitlements, risky URL schemes, third-party components, and binary-hardening characteristics may be identified without running the application [1-4]. The static analysis approach is very appealing with regard to repeatability and scale, but the available ecosystem is broken. General mobile scanners, binary parsers, reverse-engineering suites, signature engines, secret detectors, and source-oriented rule engines each expose a different slice of the security state. The resulting alerts frequently require manual correlation before an analyst can decide whether a reported condition is sufficiently evidenced to be treated as a vulnerability. MobSF provides broad mobile application analysis, while tools such as Ghidra, LIEF, YARA, Gitleaks, and Semgrep provide complementary capabilities at different abstraction levels [5-10].

Large language models (LLMs) can assist with planning, evidence synthesis, and explanation, but unconstrained model output is

not an acceptable substitute for reproducible security evidence. A persuasive natural-language explanation can still be wrong. DeepIPA therefore treats LLMs as reasoning components over tool-produced evidence rather than as vulnerability oracles. Its central design principle is to separate stochastic investigation from deterministic confirmation. The system first acquires and normalizes evidence, then allows specialized agents to investigate through controlled Model Context Protocol (MCP) tools, and finally applies machine-readable evidence contracts before a finding can be classified as "Confirmed".

The research question is: *Can a multi-agent architecture statically dissect an iOS IPA, correlate heterogeneous evidence through program/security graphs and MCP-orchestrated tools, and verify findings with evidence contracts sufficiently well to reduce false-positive confirmation while maintaining useful detection coverage?*

The current scope is deliberately static. DeepIPA does not claim to prove every runtime exploit and does not equate a static finding class with universal exploitability. Dynamic device or simulator confirmation is reserved for future work. The main contributions of this study are:

- **C1:** An end-to-end static IPA security dissection pipeline that doesn't run the app; and a normalized multi-tool evidence layer and program/security graph to connect binaries,

APIs, strings, endpoints, entitlements, components, and candidate findings:

- **C2:** A normalized multi-tool evidence layer and program/security graph that links binaries, APIs, strings, endpoints, entitlements, components, and candidate findings;
- **C3:** Multi-agent investigation with separated roles of Planner, Investigator, and Validator orchestrated by MCP; and
- **C4:** Machine-readable evidence contracts and ternary Confirmed/Rejected/Unknown decisions, which help bring the uncertainty to the fore and limit the LLM's capacity to support its confirmation;
- **C5:** Comparison of precision, recall, F1, accuracy, false positive rate, risk separation, and finding coverage scores for 25 IPA files and six comparator tools empirically.

2. Literature Review

Mobile application security testing is often broken down into standards, like the OWASP Mobile Application Security Verification Standard (MASVS), and accompanying testing guidance. The security requirements are organized into groups within MASVS, which categorizes them based on storage, cryptography, authentication, network communication, platform interaction, code quality, resilience, and privacy, and gives a helpful vocabulary for organizing static findings [1]. The behavior of ATS and the Mach-O environment of executable files it supports is also documented on the Apple platform [2][3]. There are some general-purpose mobile analysis frameworks, which provide quite a good starting point. MobSF provides an automated way to perform static and dynamic analysis on mobile applications, and it is also a popular tool for extracting metadata, reviewing configuration, and triaging vulnerabilities [5]. A wide scan output is mostly tool-local observations, however. DeepIPA is more concerned with the following correlation and verification problem:

Multiple observations are normalized, correlated, investigated, and confirmed only if they are accompanied by explicit evidence requirements.

The program evidence from binary-analysis and reverse-engineering tools adds to the depth of evidence. Ghidra can disassemble, decompile, inspect symbols, and perform cross-references on executable formats such as Mach-O [6][7], while LIEF can be used to parse and manipulate programs in formats such as Mach-O. Capstone has a lightweight multi-architecture disassembly engine [11]. These are tools that can be used to generate evidence but, by themselves, are not a multi-agent verification policy for the whole IPA package. The specialist security engines are written to detect more specific issues. Pattern-based rules for identifying binary or textual artifacts are provided by YARA, secrets are provided by Gitleaks, rule-driven static analysis by Semgrep, inventory software components by Syft, and executable capabilities by Capa. Together, they are more useful than individually—the secret-like string, the imported API, the network endpoint, and the graph path—spell the secret. This fusion can be naturally represented in program graphs. Flat finding lists are missing relationships that are captured in call graphs, control-flow graphs, and data-flow graphs. DeepIPA enriches the IPA with a security evidence graph linking the IPA, binaries, APIs, strings, endpoints, entitlements, third-party components, and candidate findings. Such relationships are modeled and queried in the prototype using the NetworkX tool [12].

The MCP is a standardized protocol that allows LLM applications to access and utilize tools that are exposed by servers. Instead of permitting an agent to have free reign on the shell, MCP allows the definition of named, schema-defined tool calls and structured results, thus enhancing the auditability and reproducibility of evidence acquisition [13]. In DeepIPA, the MCP is the control plane: the Planner will choose evidence queries based on missing fields in the contracts, and the Investigator will get back structured evidence for

analysis. The challenge of this research is at the intersection of mobile scanning, reverse engineering, graph-based evidence correlation, and security reasoning with LLMs. Many useful signals can be detected with existing tools, but there is no common confirmation mechanism in the considered toolchain, combining heterogeneous static evidence with separated agent roles and a deterministic contract gate. DeepIPA is designed to overcome this by putting LLM reasoning in the middle, between the deterministic evidence acquisition and the deterministic verification, to provide the path through which model confidence should be arrived at to determine the truth.

3. Problem Statement

This research focuses on the challenge of missing a single, auditable tool for translating heterogeneous IPA static-analysis outputs into decisions about vulnerabilities that are based on evidence. There's a risk of overlapping or incomplete signals with conventional tools, and a risk of plausible yet unbacked conclusions with the LLM analysis. The challenge is thus not to find more indicators but to correlate them, and to obtain missing evidence in an adaptive manner, to maintain the uncertainty and to ensure that the vulnerability can only be confirmed if a predefined evidence contract is fulfilled.

4. Objectives of the Study

- To create an automated tool that dissects static security of iOS IPA packages and Mach-O binaries. To combine multiple kinds of metadata, binary, secret, signature, component, and rule-based analyzers into a common evidence representation.
- To build program and security evidence graphs that aid in the investigation of potential evidence in a relationally aware manner.
- To control the use of tools by Planner, Investigator, and Validator agents using MCP. To create evidence contracts that can only return Confirmed, Rejected, and

Unknown and do not return any unsupported confirmation.

- To compare DeepIPA to a set of fixed tools that implement static analysis for app-level classification and finding coverage.

5. Methodology

5.1. Research Design and Scope

The study follows a system-design and empirical-comparison methodology. The current DeepIPA prototype performs static analysis only: it dissects the IPA archive and its packaged artifacts without launching the application. The input is an IPA file A , and the output is a set of findings $F = \{f_i\}$. Each finding contains a rule identifier, severity, taxonomy mapping, evidence bundle E_i , and a decision $d_i \in \{\text{Confirmed, Rejected, Unknown}\}$. Runtime-only behavior, server-side authorization logic, social engineering, and hardware attacks are outside the present scope.

5.2. Dataset and Ground Truth

The experimental workbook contains 25IPA files spanning clean benign applications, suspicious/medium risk applications, intentionally vulnerable laboratory applications, and the public DVIA-v2 vulnerable application. The evaluation metadata defines medium- and high-risk applications as insecure ground truth and uses a security-score threshold below 85 to classify an application as predicted insecure. Seven applications are clean at the app level, while 18 are treated as insecure (Table 1).

5.3. Deterministic Static Evidence Acquisition

The IPA is unpacked into a job workspace and the pipeline identifies the .app bundle, main executable, frameworks, property lists, provisioning files, resources, and configuration artifacts. Deterministic stages include metadata and policy inspection, Mach-O hardening checks, string and secret intelligence, cryptographic and network indicators, signature/rule engines, component inventory,

and optional reverse-engineering adapters. Candidates keep raw locations and samples; they are not automatically considered confirmed vulnerabilities.

5.4. Evidence Normalization and Program Graphs

The outputs of the heterogeneous engines are normalized into a common schema and combined with DeepIPA rule identifiers. Duplicate observations are combined to prevent double counting. The graph layer will show nodes like IPA package, binary, function/API, string, endpoint, entitlement, component, and finding. Edges define containment, reference, call, flow, or evidentiary relationships. This representation is used to represent structured questions like “Is there code that references a secret-like token to a network or authentication sink?”

5.5. MCP-Orchestrated Multi-Agent Investigation

DeepIPA separates reasoning into three roles. The Planner interprets inventory and graph context, prioritizes candidate findings, and chooses which MCP evidence tools should be called. It is explicitly prohibited from declaring a finding Confirmed. The Investigator executes evidence-oriented queries through MCP, records source/sink/path context and contradictions, and produces a structured candidate. The independent Validator re-checks the evidence against the relevant contract. This separation of duties is intended to reduce self-certification and correlated reasoning error (Figure 1).

5.6. Evidence Contracts and Decision Rule

Each static rule r declares required evidence fields and confirmation predicates in a machine-readable contract C_r . Given an evidence bundle E_i , the final decision is defined as:

$$d_i = \text{Verify}(E_i \ C_r(i)). \quad (1)$$

The Verify operation does not accept an LLM self-report as sufficient evidence. Confirmed is returned only when required evidence is complete, and predicates are satisfied; Rejected is used when evidence contradicts or fails the definition; Unknown is retained when evidence is insufficient. Unknown is therefore a scientific outcome rather than an error state.

DeepIPA targets zero false positives only on the evaluated subset of statically provable, contract-eligible classes. It does not claim zero false positives or complete detection for all iOS vulnerabilities. The restriction is mandatory, as a lot of security properties need runtime state, server behavior, user interaction, or exploitability analysis.

5.7. Evaluation Metrics

The study reports recall, precision, F1 score, accuracy, false-positive rate, and confusion count at the application level. The metrics are computed as:

$$\text{Precision} = \frac{TP}{TP+FP} \quad (2)$$

$$\text{Recall} = \frac{TP}{TP+FN} \quad (3)$$

$$F1 = \frac{\text{Precision} \cdot \text{Recall}}{\text{Precision} + \text{Recall}} \quad (4)$$

$$\text{Accuracy} = \frac{TP+TN}{TP+TN+FP+FN} \quad (5)$$

$$\text{FPR} = \frac{FP}{FP+TN} \quad (6)$$

It also compares average security scores of clean, suspicious, and vulnerable groups, the separation of risk between the clean and vulnerable groups, average findings on vulnerable applications, and the finding coverage for DVIA-v2.

6. Experimental Work

6.1. Prototype Stack

The DeepIPA prototype uses the Python programming language on Linux with a FastAPI back-end, and a research web interface is provided for upload, live worker logs, overview, findings, binary information, entitlements, ATS status, findings verified, and evidence exploration. IPA/Mach-O parsing is supported for LIEF and custom analyzers; the security evidence graph is represented by NetworkX software. Agent roles are defined as Planner, Investigator, and Validator, and communicate with servers designed specifically for this purpose (MCP servers). The results are exported to JSON and Excel format for future analysis.

6.2. Tool Layer

It contains the DeepIPA core and reverse/static analysis tools; Semgrep, Gitleaks, Syft, YARA, Capstone, capa, and string-intelligence modules are optional adapters for MobSF, Ghidra, radare2, and Rizin, as well as several other modules. MCP servers make available limited functionality like decoding plist, looking at Mach-O, queries for secrets, network evidence, graph traversal, and signature/rule evaluation. The design is such that the agents cannot execute any unrestricted shell. Comparators are interpreted in two groups. Primary IPA SAST peers are DeepIPA, IPA Auditor, MobSF, and Pavise. YARA, Gitleaks, and Semgrep are treated as niche detectors for signatures, secrets, or rules. The <85 score is an app-level triage threshold only and does not imply a shared vulnerability ontology across tools.

6.3. LLM Configuration and Agent Controls

To improve reproducibility, the laboratory configuration used for the LLM-assisted agent path is now disclosed. The Planner uses grok-4.6, the Investigator uses composer-2.5, and the Validator uses grok-4.5, accessed through the Cursor SDK with MCP tool access. The default agent timeout is 180 s, and the maximum number

of LLM-enriched findings is 5 (configurable). Temperature is left at provider defaults. These settings govern the investigation and narrative stages; they do not replace the deterministic evidence-contract gate. An important reproducibility distinction is maintained: for contract-eligible static classes, an LLM cannot upgrade an UNKNOWN result to CONFIRMED. Only the deterministic evidence-contract gate can produce CONFIRMED when the required evidence fields and predicates are satisfied. Agent-generated narrative text may vary across runs; this limitation is explicitly acknowledged for future consistency testing (Table 2).

6.4. Experimental Comparators

The experiment includes the comparison of DeepIPA with the other tools: MobSF, IPA Auditor, Pavise, YARA, Gitleaks and Semgrep [14][15]. There are various such systems, some of which are more general, mobile scanners, and some of which are more specialized, rule- or secret-detectors. The comparison is thus not for a 100% match of the internal detection semantics of an app, but for the fact that their outputs are similar within the common app-level score threshold of the workbook.

7. Results and Discussion

7.1. Overall App-Level Performance

The system that performed best in terms of recall (0.72), F1 score (0.84), and accuracy (80%) was DeepIPA, with precision of 1.00 and an app-level false-positive rate of 0. Its confusion counts were $TP=13$, $FP=0$, $TN=7$, and $FN=5$ (Table 3). MobSF was the best competitor in terms of recall and F1, with scores of 0.33 and 0.50, respectively. IPA Auditor scored F1 0.43, Pavise and YARA scored F1 0.36 each, and Gitleaks and Semgrep scored F1 0.29 each. All systems had zero app false positives at the workbook threshold, so the key difference was in the number of insecure apps they were able to detect as opposed to the number of false positives (Figure 2).

7.2. Risk-Score Separation

DeepIPA produced the strongest separation between clean and vulnerable applications. Its mean security score decreased from 96 for clean applications to 50 for vulnerable applications, a 46-point separation. MobSF produced a 28-point separation, followed by IPA Auditor (25), Pavise (20), YARA (19), Gitleaks (13), and Semgrep (6) (Table 4). The larger separation indicates that DeepIPA assigned materially lower security scores to vulnerable artifacts while retaining high scores for clean artifacts (Figure 3).

7.3. Finding Coverage on Vulnerable Applications

DeepIPA reported an average of 14.7 findings on vulnerable applications, substantially above MobSF (5.3), IPA Auditor (2.3), Pavise (2.2), YARA (2.0), Gitleaks (1.1), and Semgrep (0.8). This result is consistent with the framework design: DeepIPA combines several evidence sources rather than relying on one narrow detector. The finding count should not, however, be interpreted as an exploit count; it represents static finding classes and evidence-backed security conditions (Table 5).

7.4. DVIA-v2 Case Study

The supplied results are most clearly contrasted by DVIA-v2 [16] in a single-use application. DeepIPA gave the machine a score of 0 and found 47 issues. In this experiment, MobSF reported 12 findings, Pavise reported 3 findings, IPA Auditor reported 2, YARA reported 2, and Semgrep reported none. The case highlights the usefulness of wide evidence fusion for a purposefully vulnerable mobile application, but the exactness of the semantic interrelationships between individual findings should be approached with care, since there is variation in the rule sets used by different tools as comparator applications (Table 6). Finding-level interpretation: the DVIA-v2 count is reported as static, contract-satisfying alerts; overlapping rule hits are possible, so the 47 findings are not

claimed to represent 47 unique manually verified CWE instances (Figure 4).

7.5. Analysis Time

The recorded 0.01?4.36 s per IPA (mean 0.24 s) in the experimental workbook measures the deterministic static path only: IPA extraction/inventory, DeepIPA static analyzers, evidence-contract validation, and scoring. They exclude Planner/Investigator/Validator LLM API round-trips and multi-step MCP agent loops. Accordingly, 0.24 s is not presented as end-to-end multi-agent wall-clock time. Here, T_{static} was used to distinguish the deterministic static + contract-gate time reported in the workbook from T_{agent} , the substantially slower full multi-agent LLM path. These timings were obtained in a laboratory environment and cannot be generalized to larger production IPAs.

7.6. Interpretation and Practical Implications

The empirical pattern is that high precision and recall improved; the ranking of DeepIPA was not achieved with the acceptance of more false positives at application level, but by identifying a higher percentage of the applications labelled insecure. This is consistent with the evidence-contract design due to the fact that only evidence-complete conditions are used for confirmation, and the other cases are left undecided, as they are Unknown instead. The results also reveal the reasons behind the non-completeness of the specialist detector as a mobile security scanner. Gitleaks and Semgrep are good add-on tools for secrets and for rules, respectively, but do not have as high a recall in this app-level benchmark, likely due to their more limited scope. DeepIPA can be thought of as an orchestration and verification layer where specialist evidence can be consumed, connected via graphs, and candidate conclusions can be passed on through a common decision policy.

Further, the workbook only includes aggregate app-level ground truth, and not a complete adjudicated vulnerability-by-vulnerability reference set for the 500 planned

static finding classes. Therefore, the best defensible empirical evidence is the fact that there were no false positives when the threshold was applied to the supplied corpus at the app-level. It would be desirable to have a wider zero-false-positive claim across all classes of vulnerabilities (which would need a wider set of rules to be ground truth, rules to be labeled independently, and repeated runs, and tests on a larger external corpus), but this is beyond the scope of this work.

7.7. Ablation Design

To examine which components drive the observed gains, the revision adds an ablation ladder. A0 is a single-tool baseline using one comparator alone. A1 is a multi-tool ensemble without LLM agents. A2 is DeepIPA deterministic, combining the analyzers, evidence graph, and evidence contracts. A3 is the full system, adding the Planner, Investigator, and Validator to A2. The intended hypothesis is that A1 should increase recall relative to A0 by combining heterogeneous evidence; A2 should improve confirmation trustworthiness through graph correlation and deterministic contracts; and A3 should improve investigation and explanation without freely expanding CONFIRMED beyond the contract gate. This design also makes explicit that the LLM layer is not the source of vulnerability proof (Table 7).

8. Future Scope

Future work will extend DeepIPA from static dissection to hybrid static-dynamic verification. Unknown findings that depend on runtime behavior can be forwarded to simulator- or device-based analysis, traffic interception, and instrumentation using tools such as Frida or LLDB. Runtime evidence contracts can then preserve the same methodological principle: dynamic evidence, rather than LLM confidence, must satisfy a machine-checkable rule before confirmation. Additional work should expand the externally sourced IPA corpus, create independently labeled rule-level ground truth,

quantify run-to-run LLM variance, evaluate ablations of the Planner/Investigator/Validator roles, measure cost and tool-call counts, and assess robustness under obfuscation and stripped Swift binaries.

9. Conclusion

This study presented DeepIPA, a multi-agent framework for static security dissection of iOS IPA binaries. The framework combines deterministic static engines, normalized program/security graphs, MCP-controlled evidence acquisition, separated Planner–Investigator–Validator roles, and deterministic evidence contracts. This architecture constrains LLM hallucination at the point that matters most: final vulnerability confirmation. In the 25-IPA evaluation, DeepIPA achieved precision of 1.00, recall of 0.72, F1 of 0.84, 80% accuracy, and zero app-level false positives under the defined threshold. It also produced the strongest clean-to-vulnerable risk separation and substantially higher finding coverage on vulnerable applications, including DVIA-v2. The findings indicate that evidence-gated multi-agent orchestration can improve static mobile security coverage without requiring unsupported model confidence to be treated as proof. The current system remains a static-analysis framework, and its zero-false-positive objective must remain scoped to evaluated, statically provable contract classes.

DECLARATIONS

Copyright Permission: All figures and tables used in the manuscript are the authors' own work and have not been reproduced from any copyrighted source. We gave the form of copy right to you as well. Copy right form was also shared with you before.

Funding Source: No specific external research grant or project funding was identified for this manuscript.

Data Availability: Data will be available upon request.

Use of AI and AI-Assisted Technologies: AI-assisted tools were used during manuscript preparation for language improvement, organization, and drafting support. The scientific claims, experimental data, interpretation, references, and final conclusions remain the responsibility of the authors.

Conflict of Interest: The authors declare no conflict of interest regarding the publication of this paper.

REFERENCES

- [1] OWASP Foundation, “Mobile Application Security Verification Standard (MASVS),” *OWASP Mobile Application Security*, 2026. <https://mas.owasp.org/MASVS/>
- [2] Apple Inc., “Preventing insecure network connections,” *Apple Developer Documentation*. <https://developer.apple.com/documentation>
- [3] Apple Inc., “Mach-O programming topics,” *Apple Developer Documentation Archive*. <https://www.cs.miami.edu/home/burt/learning/Csc521.091/docs/MachOTopics.pdf>
- [4] MITRE Corporation, “Common Weakness Enumeration (CWE).” <https://cwe.mitre.org/>
- [5] MobSF Project, “Mobile Security Framework (MobSF),” *GitHub repository and documentation*. <https://github.com/MobSF/Mobile-Security-Framework-MobSF>
- [6] National Security Agency, “Ghidra Software Reverse Engineering Framework,” *GitHub repository*. <https://github.com/NationalSecurityAgency/ghidra>
- [7] R. Thomas, “LIEF – Library to Instrument Executable Formats,” <https://www.romainthomas.fr/publication/lief/>
- [8] VirusTotal, “YARA: The pattern matching swiss knife,” *GitHub repository*. <https://github.com/VirusTotal/yara>
- [9] Gitleaks, “Gitleaks: Find secrets with Gitleaks,” *GitHub repository*. <https://github.com/gitleaks/gitleaks>
- [10] Semgrep, “Semgrep Static Analysis,” *Documentation and repository*. <https://github.com/semgrep/semgrep>
- [11] N. A. Quynh, “Capstone: Next-generation disassembly framework,” *Black Hat USA*, 2014. <https://blackhat.com/docs/us-14/materials/us-14-NguyenAnh-Capstone-Next-Generation-Disassembly-Framework.pdf>
- [12] A. A. Hagberg, D. A. Schult, and P. J. Swart, “Exploring network structure, dynamics, and function using NetworkX,” in *Proc. 7th Python in Science Conf. (SciPy2008)*, Pasadena, CA, USA, 2008, pp. 11–15.
- [13] Model Context Protocol, “Model Context Protocol Specification,” 2026. <https://modelcontextprotocol.io/specification/2026-07-28>
- [14] P. Gianchandani, “Damn Vulnerable iOS App (DVIA),” intentionally vulnerable iOS application benchmark. <https://github.com/prateek147/DVIA-v2>
- [15] The Cyber Sandeep, “IPA Auditor,” *GitHub repository*. <https://github.com/thecybersandee/ipaauditor>
- [16] A. Mutlugun, “Pavise,” *GitHub repository*. <https://github.com/ahmetmutlugun/pavise>

Tables

Category / Risk label	Number of IPA files
Clean	7
Vulnerable	9
Suspicious	9

Table 1: Composition of the evaluated IPA corpus.

MCP capability	Primary evidence returned
IPA/Core	Bundle inventory and extracted artifact metadata
Plist	Info.plist, ATS, URL schemes, and configuration values
Mach-O	Segments, symbols, imports, hardening, and binary properties
Secret	Secret candidates, locations, and classification context
Network	Endpoint and protocol evidence
Graph	Paths and neighborhoods linking evidence to candidate findings
Rule engines	YARA/Semgrep-style rule results and locations

Table 2: Representative MCP capabilities and returned evidence.

Tool	Precision	Recall	F1	Accuracy (%)	FPR	TP	FP	TN	FN
DeepIPA	1.00	0.72	0.84	80	0.00	13	0	7	5
MobSF	1.00	0.33	0.50	52	0.00	6	0	7	12
IPA Auditor	1.00	0.28	0.43	48	0.00	5	0	7	13
Pavise	1.00	0.22	0.36	44	0.00	4	0	7	14
YARA	1.00	0.22	0.36	44	0.00	4	0	7	14
Gitleaks	1.00	0.17	0.29	40	0.00	3	0	7	15
Semgrep	1.00	0.17	0.29	40	0.00	3	0	7	15

Table 3: App-level performance of DeepIPA and comparator tools.

Tool	Clean avg.	Suspicious avg.	Vulnerable avg.	Risk separation
DeepIPA	96	82	50	46
MobSF	100	88	72	28
IPA Auditor	100	94	75	25
Pavise	100	95	80	20
YARA	100	98	81	19
Gitleaks	100	99	87	13
Semgrep	100	100	94	6

Table 4: Average security scores and clean-to-vulnerable risk separation.

Tool	Avg findings – Clean	Avg findings – Suspicious	Avg findings – Vulnerable
DeepIPA	2.0	4.8	14.7
MobSF	0	2.3	5.3
IPA Auditor	0	0.6	2.3
Pavise	0	0.6	2.2
YARA	0	0.2	2.0
Gitleaks	0	0.1	1.1
Semgrep	0	0	0.8

Table 5: Average findings by application category.

Tool	Security score	Findings	Workbook interpretation
DeepIPA	0	47	Proposed system—strongest detection on DVIA
IPA Auditor	84	2	Partial / weak detection
MobSF	88	12	Partial / weak detection
Pavise	80	3	Partial / weak detection
YARA	84	2	Partial / weak detection
Gitleaks	100	0	Missed almost all issues (score stays high)
Semgrep	100	0	Missed almost all issues (score stays high)

Table 6: DVIA-v2 results across evaluated tools.

ID	System	Description
A0	Single-tool baseline	One comparator alone
A1	Ensemble LLM) (no	Multi-tool merge without agents
A2	DeepIPA deterministic	Analyzers + graph + evidence contracts
A3	DeepIPA full	A2 + Planner / Investigator / Validator

Table 7: Ablation configurations used to evaluate the contribution of DeepIPA components.

Figures

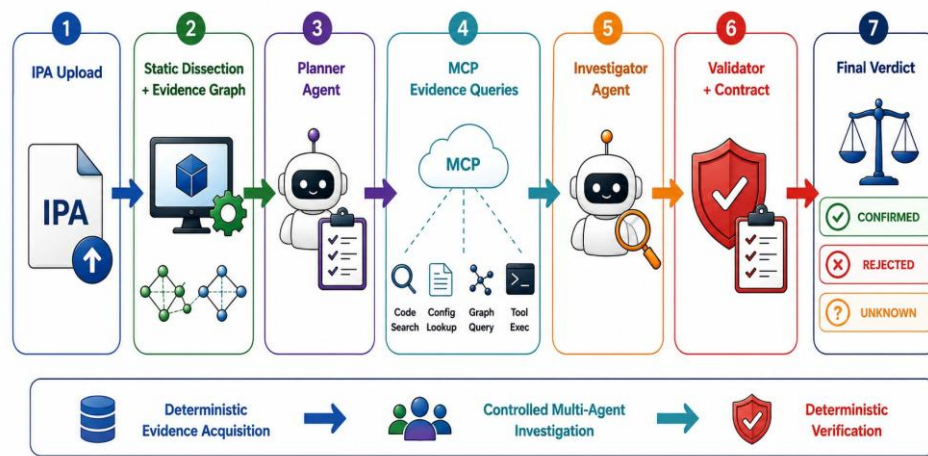


Figure 1: DeepIPA end-to-end architecture: deterministic evidence acquisition, MCP-controlled multi-agent investigation, and contract-gated verification.

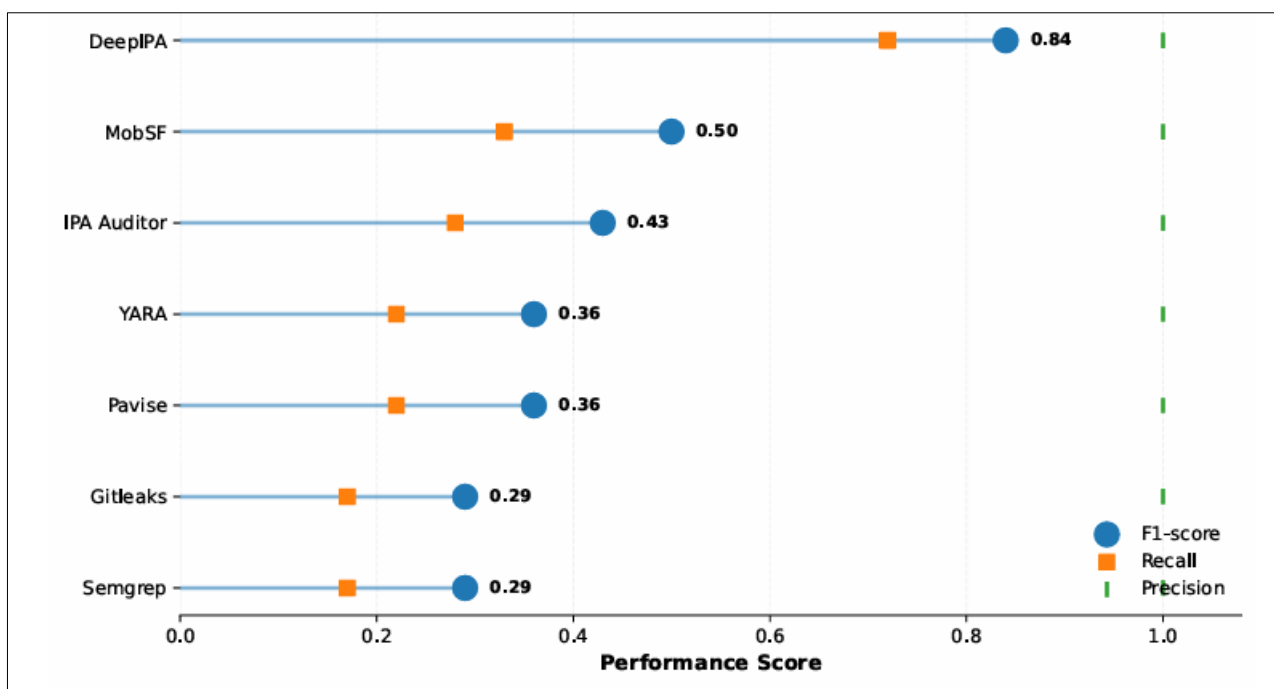


Figure 2: Performance Ranking of DeepIPA and Comparator Tools: Precision, recall, and F1 score across DeepIPA and comparator tools.

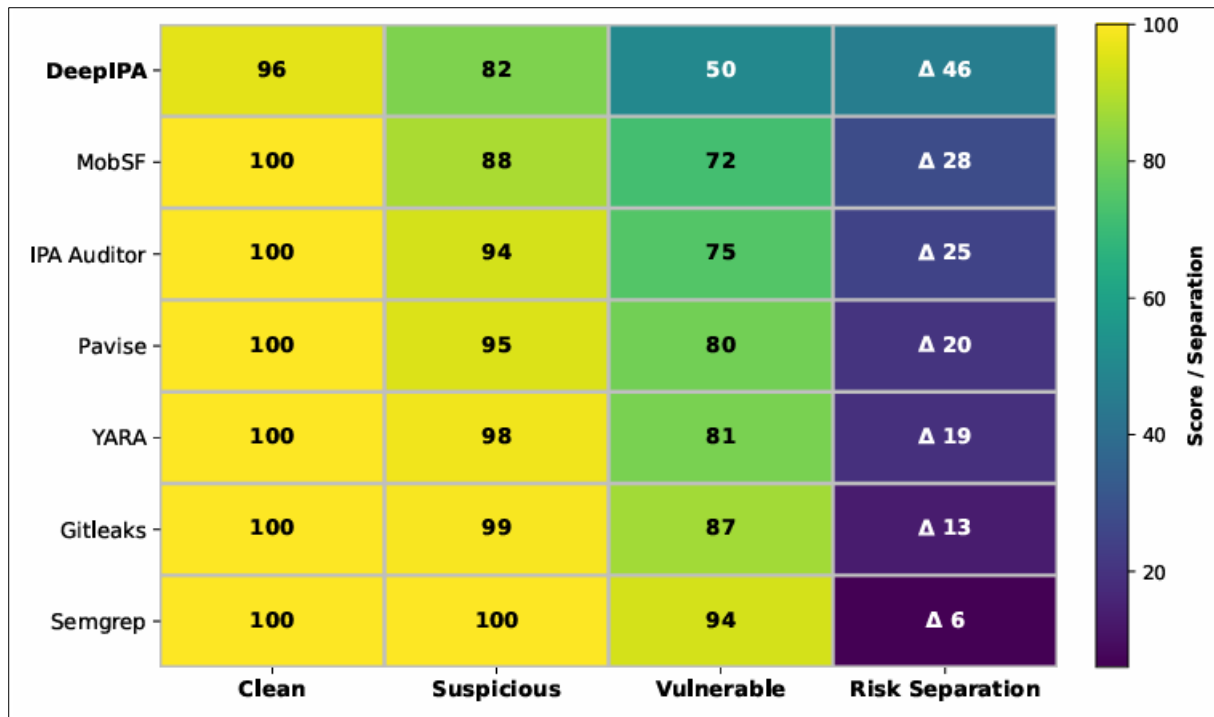


Figure 3: Security-Score Discrimination Across Application Risk Groups: Average security scores by application risk group (higher score indicates a safer-looking application).

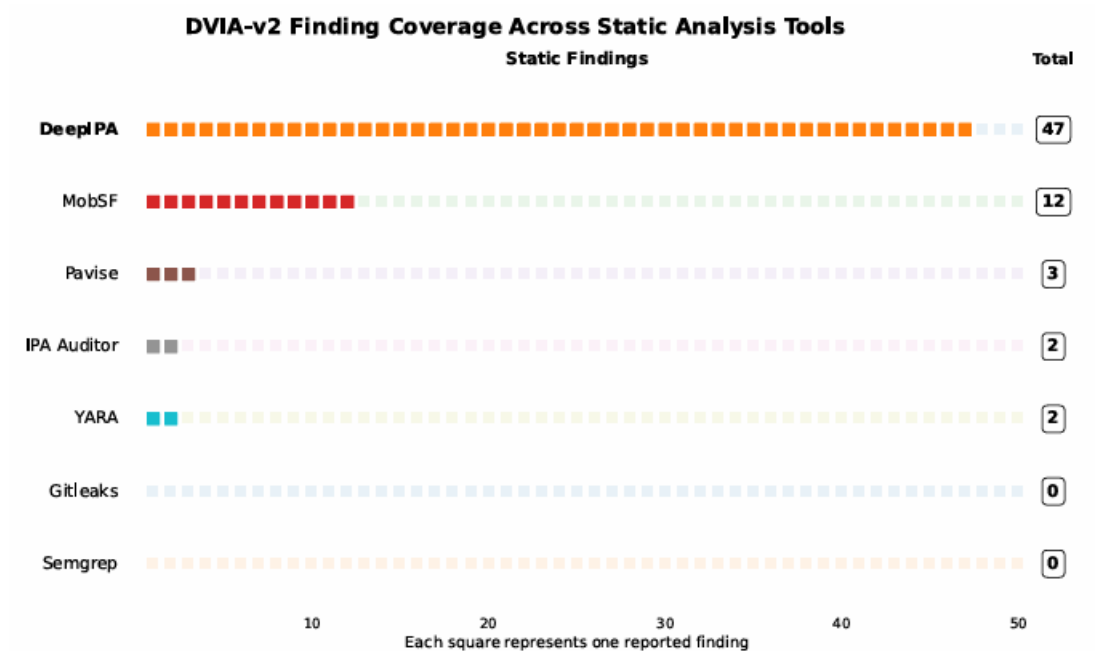


Figure 4: Number of findings reported for DVIA-v2 by the evaluated tools.