

Component Level Security Analysis for MCP Servers Agent Skills and Plugins

A technical overview of BlueRock's
Agentic Trust Registry and its
assessment method



Executive summary

AI agents introduce security risks at the boundary between code, instructions, tools, and delegated authority. An MCP server may expose powerful operations. A skill may supply instructions the model follows. A plugin may add capabilities that are difficult to infer from a repository-wide grade. These components can change independently, and a component that appears safe by code-pattern analysis may still influence an agent in unsafe ways.

Security assessment therefore needs to answer questions at the component level: what does each component do, what can it access, what instructions does it give an agent, and what evidence supports a vulnerability finding?

BlueRock's Agentic Trust Registry (ATR) applies this model to MCP servers, skills, and plugins. ATR combines deterministic static analysis with rules written for agent-specific risks. Its analysis engine evaluates each component against applicable rules, requires evidence for vulnerable verdicts, and sends the first assessment through a cross-vendor review and reconciliation process. A report is delivered as complete only after all required

stages succeed.

ATR also produces a trust score and tier to help developers and security teams prioritize review. The score is a triage aid, not a probability of compromise or a guarantee that a component is safe. Findings, code evidence, capability information, and coverage limits remain the basis for technical decisions.

ATR brings component-level security analysis to a substantial agentic software ecosystem. As of October 14, 2026, BlueRock's published registry metrics report more than 14,000 MCP servers assessed and more than 185,000 tools inventoried. BlueRock also reports 92% detection of known agent attacks in its head-to-head benchmark. These figures describe different dimensions of the system: assessment reach, capability visibility, and performance on a defined test set. This paper explains the assessment method behind ATR: component-specific rules, code-level evidence, cross-vendor challenge review, and validated reports that help teams decide what to adopt, investigate, and remediate.

1. The assessment problem is changing

Conventional application security methods remain useful for agentic software. Static analysis can detect known code patterns; dependency analysis can identify vulnerable packages; secret scanners can find exposed credentials. Those methods should remain part of the assessment.

The gap is that agentic systems add security-relevant behavior that may not be represented by conventional code patterns alone:

Instructions can steer execution. Tool descriptions, manifests, and skill instructions are read by a model. A malicious or misleading instruction can shape which actions the model takes.

Tools inherit authority. A tool may operate with a user's credentials, files, network access, or other permissions. Its capability and blast radius matter alongside code quality.

Components can change after approval. Tool definitions and packages may be updated, fetched dynamically, or replaced by lookalike publishers. A previous review may no longer describe the component in use.

Agent-specific protocol risks matter. For example, a local MCP server that does not validate the requesting origin and host may be exposed to DNS rebinding.

Repository-level summaries can hide component-level risk. A single repository can contain many MCP servers, skills, plugins, and application code. One aggregate result can obscure which component has a finding or which capability it exposes.

These observations do not mean every agent component is unsafe or that traditional scanners have no value. They mean an assessment intended for agentic software should preserve code-level checks while also analyzing instructions, component identity, tool capabilities, and agent-specific control paths.

Assessment unit: the component

ATR treats a separately identifiable component as the primary unit of assessment. One repository can therefore produce multiple component reports. Each report identifies the rules applied, the findings and evidence for that component, and the capabilities exposed by its tools.

Components are identified from manifests and code evidence, including skill files such as SKILL.md, plugin manifests, MCP configuration, and implementation signals. A directory can represent more than one component. When a repository root functions as both a skill and an MCP server, ATR registers both and applies both rule sets.

The current dedicated rule sets cover MCP servers, skills, and plugins. Other agent application types may be identified and evaluated under common rules, but dedicated rule sets for agent-framework applications and retrieval-augmented generation pipelines are not part of the current assessment.

Example: risk in one of six components

An open-source browser-automation agent contains six components: one MCP server and five skills that support the project's own development workflow. ATR assessed each one separately. The five skills had no vulnerable findings of their own. The MCP server had three, and the clearest is a high-severity finding under ATR-M0003, origin and host validation.

The MCP server includes a local bridge that accepts WebSocket connections on localhost and never checks the Origin header. A browser lets any web page open a WebSocket connection to any host, so only the server's own Origin check can refuse one, and this bridge has none. A page that connects before the agent's browser extension does is accepted as the extension, and the bridge then sends it the next task, including the user's LLM API key. The report cites four code locations as evidence, each with a short excerpt.

1. The assessment problem is changing

The finding tells a developer what to change: reject connections from any origin other than the extension's, or require a per-session secret that the extension presents when it connects. It tells a security reviewer where the risk sits: in the MCP server, not in the five skills, which do not need the same remediation.

A second example involves agent instructions. A large open-source workflow-automation project contains 40 skills. One of them, which manages stacked pull requests,

tells the agent always to run its merge command with the flag that skips the confirmation prompt, so the agent can merge an entire stack of pull requests without asking the user. ATR reported this under ATR-P0002, consent suppression, at critical severity. None of the other 39 skills had a consent-suppression finding. The fix is to drop that flag from the merge instruction, so the user confirms before anything is merged, and to keep non-interactive flags for read-only commands.

2. What ATR evaluates

Rules applied by component type

ATR uses 34 versioned rules organized into four namespaces. Common rules and a guardrail apply to every

repository; component-specific rules apply to each detected MCP server, skill, or plugin.

Rule namespace	Count	Applied to
Common	14	Every repository
Guardrails	1	Every repository
MCP server	11	Each MCP server
Skills and plugins	8	Each skill or plugin

An MCP server is assessed against 26 rules: 14 common, one guardrail, and 11 MCP-specific rules. A skill or plugin is assessed against 23 rules: 14 common, one guardrail, and eight component-specific rules. If a component satisfies

more than one type, it receives each applicable rule set.

Examples of agent-specific risks addressed by the current rules include:

Risk	Example of what the rule evaluates
Tool definition integrity	Whether a tool definition can change after approval without integrity controls or renewed approval
Tool name spoofing	Whether a tool name collides with or imitates another tool
Prompt or metadata injection	Whether tool descriptions contain instructions that could steer a model
Origin and host validation	Whether a local server validates requests to reduce DNS rebinding exposure
MCP authorization	Whether remote authorization follows the expected OAuth and PKCE protections
Skill credential harvesting	Whether a skill reads credentials or conversation content and transmits it externally
Consent suppression	Whether instructions attempt to bypass confirmations or human review
Declared versus actual behavior	Whether a skill's behavior exceeds its declared purpose
Package provenance and mutation	Whether a skill or plugin can be impersonated or modified after approval

2. What ATR evaluates

Common rules address security weaknesses that also matter when a model is in the loop, including command injection, server-side request forgery, static secrets, dependency risk, path traversal, insecure deserialization, CI/CD script injection, denial-of-wallet controls, and unsafe tool output.

Tool inventory and capability

Each component report includes an inventory of the tools that component exposes. Tools are described and assigned capability tags such as read-only, write, broad network access, secrets access, destructive, or execution-capable. These tags inform the component's capability band.

ATR scores each component and, within it, each tool. The component score is the primary result; per-tool scores

show which tools drive it.

Framework references

ATR rules map to MITRE Common Weakness Enumeration (CWE) identifiers. Agent-specific rules also reference sources such as the OWASP Top 10 for LLM and Generative AI Applications, MITRE ATLAS, and the Model Context Protocol security guidance. General code rules reference OWASP application, API, and CI/CD security material.

These mappings show the sources used in rule design. They do not mean ATR is certified against, compliant with, or exhaustive of any framework. ATR does not claim complete coverage of the OWASP LLM Top 10.

3. How an assessment works

ATR uses deterministic analysis and model-assisted code review in a staged process. Deterministic tools provide repeatable baseline checks. The model-assisted stages

analyze context, component behavior, and agent-specific rules. Validation gates check that the submitted results meet the expected schema and evidence requirements.

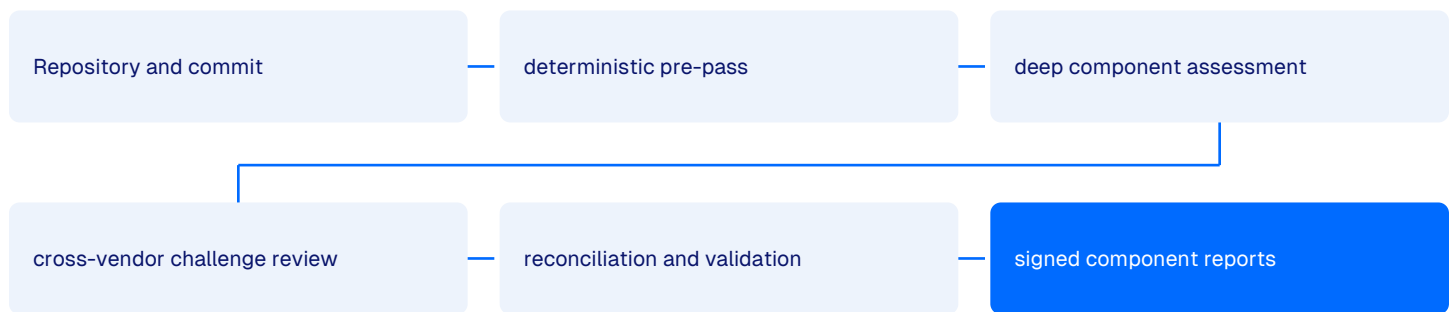


Figure 1. Assessment pipeline

Stage 0: deterministic pre-pass

Before a model reviews the repository, ATR runs a fixed suite of open-source analysis tools. The current pre-pass includes Semgrep, Trivy, Gitleaks, Bandit, and gosec. Language-specific tools run when the corresponding language is present. The tools run outside the model's working loop, and their results are supplied as a baseline.

The report records which tools ran and how many results each returned. This lets reviewers distinguish deterministic tool output from model-generated analysis.

Stage 1: deep assessment

An AI agent evaluates each component against each applicable rule. It submits findings through structured interfaces: a known rule ID, a closed-set verdict, and structured evidence. The assessment service validates submissions, rejects unknown rule IDs, de-duplicates repeated findings for the same component and rule, and checks quoted evidence against the repository files.

Quality gates check component and rule coverage and limit undecided verdicts. A pass that does not meet the

gates is retried and may be escalated to a different analysis engine.

Stage 2: cross-vendor challenge review

A second AI agent re-examines the repository and the first assessment. By default, the reviewer uses a model from a different vendor. It receives the first pass's findings, so this is a challenge review rather than a blind, independent scan. The reviewer runs its own deterministic pre-pass, reads the code, and can confirm a finding, dispute one, or raise a vulnerability the first pass did not flag.

Across 136 production assessments created from August 25 through October 9, 2026, the review proposed 643 vulnerability findings not raised in the first pass and proposed withdrawing 219 findings. Reconciliation accepted 643 of 645 proposed additions and 183 of 219 proposed withdrawals. There is no ground truth in this dataset, so these figures show how the review changed the results; they do not establish that every change was correct.

3. How an assessment works

Stage 3: reconciliation

A reconciliation step applies the review's proposed changes and resolves disagreements. It uses a model from the same vendor as the first pass, so it is not a third independent reviewer. If a finding from the first pass is omitted during reconciliation, the pipeline carries it forward and marks it rather than silently dropping it. Each component report is checked against the report schema before delivery; a report that fails validation causes the assessment to fail.

Completion and scale

An assessment is marked complete only when the deep assessment, challenge review, and reconciliation all succeed. If a stage fails after retries, the assessment is marked failed with an explanation; ATR does not label an unreviewed or partial result as complete.

The largest completed production assessment to date scored 192 components in one repository. Large repositories are divided into batches of components that can be assessed in parallel, while repository-wide rules are evaluated once.

4. Rules designed for evidence

Each ATR rule is versioned and maintained in version control. A rule includes its identifier, severity, CWE mapping, evaluation rubric, vulnerable and safe examples, required evidence, false-positive guards, context limits, and supporting references. Findings record the rule version used so that results can be compared across assessments.

Rules are written to distinguish a weakness from code that merely resembles one. For example, an evaluation rubric can require a reachable attacker-controlled input, a path to a dangerous operation, and an impact—not just the presence of a risky-looking function.

Verdicts and evidence locations

For each rule and component, the assessment returns one of four verdicts:

Vulnerable: evidence supports the rule’s defined weakness.

Secure: the evaluated behavior does not meet the rule’s vulnerable condition.

Not applicable: the component’s architecture does not expose the threat described by the rule.

Unknown: evidence is insufficient to make a supported decision.

A missing security control is treated as a possible vulnerability, not automatically as “not applicable.” A vulnerable finding contains one or more repository paths and line ranges, a short excerpt, a rationale, remediation guidance, a confidence level, and a plain-language explanation of why the issue matters.

Quoted evidence is checked against the files. If line positions have shifted, ATR attempts to correct the location. If the excerpt cannot be found, confidence is lowered and the report explains why. A vulnerable verdict that lacks evidence required by its rule after two attempts is recorded as unknown, and the report identifies the missing evidence.

Evidence classes

Rules can require evidence from several classes, depending on the weakness:

Evidence class	Question it addresses
Attacker-controlled source	Can an attacker supply the relevant input?
Execution reachability	Does the relevant code path run?
Dangerous sink	Can the operation cause harm?
Source-to-sink flow	Is there a path from the input to the operation?
Mitigation analysis	Is a guard present, and does it hold?
Attacker trigger capability	Can an attacker cause the path to execute?
Security impact	What could happen if the behavior is exploited?

4. Rules designed for evidence

A command-injection rule can require evidence across all seven classes. A rule about an absent control, such as missing transport security, may require fewer. The rule defines what is sufficient; the assessment cannot substitute a generic assertion for required evidence.

Example: ATR-M0003, origin and host validation

Threat: a web page open in the user's browser uses DNS rebinding to send requests to an MCP server on the user's machine.

Vulnerable when: an HTTP or SSE MCP listener accepts requests without checking the Origin header (and the Host header, if it listens beyond loopback), and a rebound page would gain something of the user's: a tool with file, process, network, or credential access; a listener

reachable beyond loopback; or a response that carries a credential.

Required evidence: execution reachability, mitigation analysis, attacker trigger capability, and security impact. Severity: high. CWE-942 and CWE-346.

Safe lookalike: a listener with no Origin check that listens on loopback only, whose tools work only on the caller's own input and return no credentials. A rebound page gains a session of its own and nothing of the user's, so the false-positive guard grades it secure and records the missing check as hardening.

Code shape: a configuration that allows every origin with credentials, such as `cors({ origin: "*", credentials: true })`, is flagged; an explicit list of allowed origins is not.

Resulting finding: the MCP server in the section 1 example.

5. Severity, capability, and trust scores

Inherent and realized severity

Each rule defines an inherent severity from Low to Critical. A vulnerable finding can also receive a context-adjusted realized severity. In the current method, three contextual signals can reduce severity when the rule permits them:

Transport gating: for example, a transport-specific risk may not apply when a component runs only locally over standard input and output.

Declared behavior: the behavior may fall within what the component openly says it does.

Mitigation present: a code-level guard may limit the risk.

Transport gating is derived from the rule and detected component transports. The two model-judged context signals receive credit only when both the first pass and challenge reviewer identify them. Relief is capped, and some rules refuse particular signals. A finding remains vulnerable when its realized severity is lowered; the context changes urgency, not whether the weakness is reported.

In production findings created since August 25 and completed after September 11, 2026, 375 of 2,581 vulnerability findings were reported at a lower, context-adjusted severity. In 374 of those cases, both the first pass and reviewer identified the same mitigating context. This is a description of ATR's scoring behavior, not proof that the adjustments predict real-world exploitability.

Capability is separate from trust

Each component receives a capability band—minimal, moderate, high, or extreme—based on its most consequential tool capability. A component that can execute arbitrary code, for example, may have an extreme capability band. The band is not a quality grade and does not lower the trust score by itself.

Separating capability from trust avoids treating a powerful component as defective simply because it is powerful. Capability helps reviewers understand potential blast radius; findings describe weaknesses in the component.

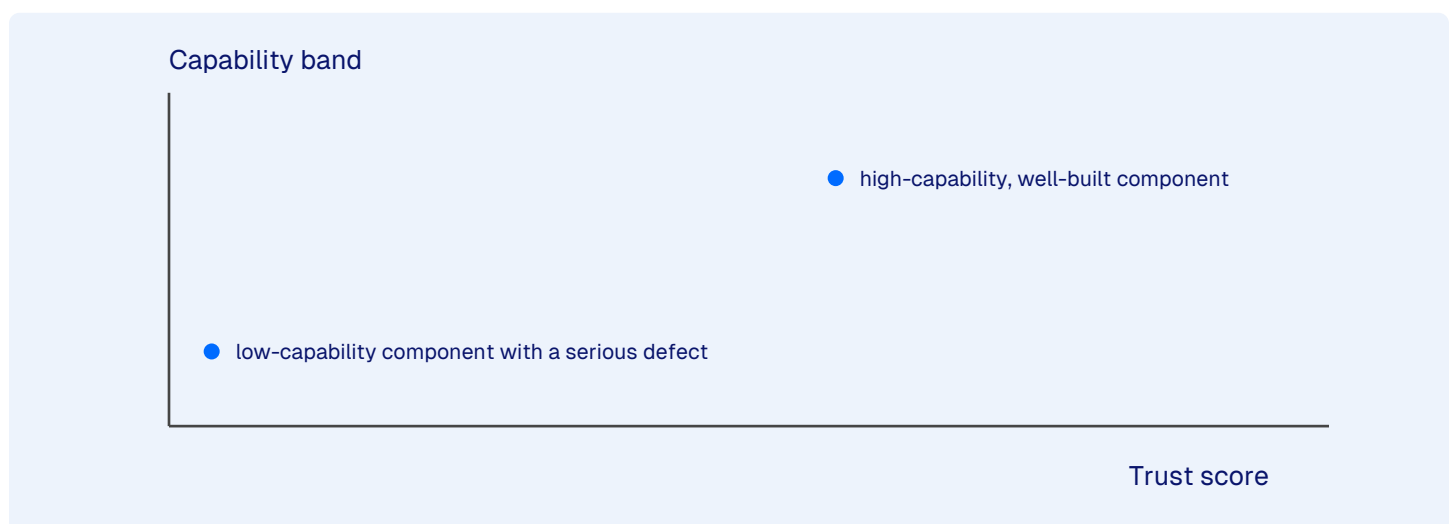


Figure 2. Trust and capability are different dimensions

5. Severity, capability, and trust scores

Trust score as a triage aid

ATR's 0–100 trust score summarizes confirmed findings to support prioritization. The method weights findings by severity and component context, accounts for corroborated context relief, and applies a curve so early serious findings have greater effect on the score. Findings that share one root defect are counted once. A confidence interval widens when evidence confidence is low, assessment coverage is incomplete, or the two analysis passes disagree; uncertainty does not increase the score. For policy decisions, the lower bound is the conservative reference.

The score is not a probability, a certification, or a substitute for examining findings. Two assessments of the same commit can differ because model-assisted analysis is not perfectly deterministic. The confidence interval is intended to communicate uncertainty around a given score.

Scores are versioned: BlueRock records the version of the scoring method used for each assessment, and registry entries show it. The method's numeric constants and tier cutoffs have not yet been calibrated against an expert-ranked set of repositories, and they may change when they are. Reassessment shows the current variance. Across 139 pairs of assessments of an unchanged commit in 73 repositories, measured in BlueRock's development environment in October 2026, the repository-level score moved by a median of 2 points, by 17 points at the 90th percentile, and by up to 80 points. Between the two runs, 14% of components and 42% of tools landed in a different tier, and the repository's overall tier changed in 22% of pairs.

The current tier labels are Strong, Adequate, Weak, Failing, and Blocked. Blocked is a separate guardrail status for an artifact identified as malicious by design; it is not a score band and is not synonymous with a low score. In the current scoring version, the cutoffs are 85 for Strong, 65 for Adequate, and 40 for Weak; a score below 40 is Failing.

6. Deterministic analysis and repository context

ATR combines analysis methods because deterministic tools and model-assisted review answer different questions. Pattern tools provide repeatable checks for known classes of issue. Model-assisted review can reason about intent, control flow, and component context. Neither method removes the need to inspect the evidence.

In addition to the static-analysis pre-pass, ATR:

computes component capability bands in code from tool capability tags;

records repository provenance signals, including publisher, signed-build or provenance indicators, known vulnerable dependencies, and maintenance activity; and

uses a deterministic pre-scan to identify changes that should trigger a full reassessment.

Repository provenance is reported separately from the trust score. It does not change the component score.

7. Report integrity and source handling

Signed component reports

ATR produces a self-contained report for each assessed component so a downstream policy engine can act on that component's result. Reports are cryptographically signed using ML-DSA-87, a post-quantum signature algorithm standardized in FIPS 204. When a customer submits code encrypted and supplies a return key, the reports are also encrypted to that key. Customers who receive reports in their own storage also receive BlueRock's public key and an offline verifier, so they can check each signature themselves.

A signature can show that report content was produced by BlueRock and has not changed since signing. A signature alone does not show that the report set is complete or that it is the latest assessment.

Excerpt masking

Reports include short evidence excerpts, which may contain secrets or personal data. The masking system is designed to identify common secret formats, credential-bearing URLs, values assigned to secret-named keys, scanner-identified secrets, and email addresses. Matching values are replaced with [REDACTED]. Masking is applied when findings are stored and again when reports are assembled. It can also mask public values, and it cannot guarantee detection of every opaque secret. Masking has run in production since October 15, 2026. Reports archived before that rollout were not re-masked and are labelled as unmasked.

How source code is handled

The description below reflects BlueRock's production environment as of October 2026.

In transit: archives submitted through the platform are encrypted by the client to a key-agreement key whose

private half remains in a cloud key-management service. Public repositories can be submitted by URL and cloned directly.

At rest: an uploaded archive is decrypted on receipt and stored with storage-level encryption so pipeline stages can read it. The default retention period is seven days. An archive associated with a failed assessment may be retained for up to one year for investigation. A public repository submitted by URL is cloned directly by each analysis stage, and no archive of it is stored.

During analysis: each analysis task uses temporary working storage for one assessment at a time. The working copy is checked for integrity before reuse in the same assessment and deleted before the task handles another assessment. Leftover processes are terminated.

Model providers: analysis agents send the repository content they read to commercial model APIs. By default, the first pass and reconciliation use Anthropic models and the challenge review uses an OpenAI model, reached through BlueRock's model gateway.

Working transcripts and static-analysis output: analysis transcripts can include file contents read by the agents, and raw static-analysis output can include secret values that the secret scanner detected. Both are kept for 30 days for customer submissions, and for one year when an assessment fails. For public repositories assessed for the public registry, they are kept indefinitely. Neither is included in customer reports.

Reports: reports contain findings, short evidence excerpts, and metadata—not a copy of the repository source. Stored report archives are kept for 90 days for customer submissions and indefinitely for public registry reports. Findings and their excerpts are also kept in BlueRock's database, which has no automatic age-based deletion today; an assessment's records can be deleted on request.

8. Keeping assessments current

Agent components and their rules change. ATR versions rules in source control, records the rule version used for each finding, and records the exact repository commit assessed.

The rule set grew from 22 rules in April 2026 to 34 by August 2026. Since July 1, 2026, rules were updated 25 times: the 24 rules present on July 1 were revised, and 10 new rules were added. These are counts of rule changes, not a claim that every change corresponds to a newly discovered attack.

For public repositories ATR monitors, changed code can trigger reassessment, up to approximately once a month

per repository. When a repository is submitted again, a deterministic pre-scan compares changes against security-relevant patterns, including authentication code, dependencies, and new source files. If no material change is detected, ATR may return the prior report. A material code change, a rule update, or an assessment older than the configured age limit can force a full reassessment. The default age limit is 30 days.

Because reports include both the assessed commit and rule versions, changes between reports can be traced to source changes, rule changes, or both.

9. Assessment scale and reported results

ATR evaluates agentic software at the level where capabilities and risks are introduced: individual components and their tools. Its reports connect identified weaknesses to code evidence, explain the capabilities a

component exposes, and provide remediation guidance.

BlueRock’s published Agentic Trust Registry metrics, dated October 14, 2026, summarize assessment reach and reported benchmark performance:

Measure	Published result
MCP servers assessed	14,000+
Tools inventoried	185,000+
Known agent attacks detected in BlueRock’s head-to-head benchmark	92%

The assessment and inventory totals describe the reach of BlueRock’s scanning. The benchmark percentage describes reported detection within a defined test set. These measures have different populations and denominators and should not be combined into a single measure of accuracy or coverage.

From scale to actionable evidence

Assessment volume is only useful when teams can act on the result. ATR identifies individual components, applies the rules relevant to their type, inventories tool capabilities, and ties vulnerability findings to repository evidence. Cross-vendor challenge review and reconciliation examine the initial findings before an assessment is delivered as complete.

This gives developers and security teams a concrete basis for review: which component contains a weakness, what that component can access or execute, where the supporting evidence appears, and what remediation is

recommended.

Scope and interpretation

The operational examples elsewhere in this paper come from separately dated assessment cohorts. Their repository, component, tool, and review counts describe those specific studies; they are not the denominators for the published registry totals above.

The 92% figure is BlueRock’s published benchmark result, not a detection rate independently established by this paper. Benchmark performance depends on the attack set, tested versions, configuration, and evaluation method. It should not be interpreted as a universal detection rate, a false-positive rate, or a guarantee that an assessed component is safe.

Trust scores support prioritization. Findings, code evidence, capability information, and assessment coverage remain the basis for technical decisions.

9. Assessment scale and reported results



BlueRock published registry metrics, October 14, 2026.

Different measurement populations; benchmark performance is specific to the tested attack set and conditions.

Figure 3. ATR assessment scale and reported benchmark performance

10. Conclusion

Agentic software changes the security assessment problem by putting instructions, tools, identity, and delegated authority in the same execution path. Repository-level summaries and conventional code checks remain useful, but a complete review of agentic software also needs to account for the distinct components that shape agent behavior and the capabilities those components expose.

ATR applies that approach to MCP servers, skills, and plugins. It combines deterministic analysis with component-specific rules, requires evidence for vulnerable verdicts, uses a cross-vendor challenge review, and reports complete assessments only after reconciliation and validation. It records capability

separately from trust and preserves commit and rule-version information to support reassessment over time.

ATR connects assessment scale with the evidence required to make component-level decisions. Developers can identify what needs to change; security teams can examine the capabilities and weaknesses that matter for adoption; and subsequent assessments can be compared using recorded commits and rule versions. The trust score helps prioritize that work. The lasting value is the traceable relationship between a component, its capabilities, the findings against it, and the evidence supporting each finding.

Appendix A. Glossary

Term	Meaning
Agentic software	Software an AI agent calls, reads, or follows, including MCP servers, skills, plugins, and agent applications.
Assessment	A complete run of the required stages against a specific repository commit.
Capability band	A code-computed summary of the most consequential capability exposed by a component's tools.
Component	A separately assessed unit inside a repository, such as an MCP server, skill, or plugin.
Confidence interval	A range around a trust score that widens with lower evidence confidence, incomplete coverage, or reviewer disagreement. It is not a statistical guarantee unless separately validated.
Realized severity	Severity after permitted, corroborated context is taken into account.
Rule severity	The inherent seriousness assigned to a weakness by its rule.
Tier	The score band or separate guardrail status associated with a component assessment.
Tool inventory	The component's tools and their capability tags.
Trust score	A 0–100 summary of confirmed findings, intended to support triage and prioritization.

Appendix B. Rule catalogue (October 9, 2026)

ATR contains 14 common rules, one guardrail rule, 11 MCP server rules, and eight skills/plugin rules. The complete rule IDs and titles are listed below; severity and CWE mappings should be included in a technical appendix or online rule catalogue after technical review.

Common rules: Unrestricted Network Fetch / SSRF; Static Secrets; Runtime Command Injection via Process Execution; DoS Protections; Logging Hygiene and Audit Trail; Container/Runtime Hardening; Dependency Risk and Supply Chain; Input Validation; Insecure Deserialization; Output Encoding and Sanitization; Path Traversal and Symlink Bypass; Denial of Wallet / Cost Controls; Tool Output Sanitization and Injection Containment; CI/CD Workflow Script Injection.

Guardrail: Weaponization—Malicious-Purpose Artifact.

MCP server rules: Tool Definition Integrity and Versioning; Unrestricted Tool Permissions; CORS / Origin / Host Validation; Binding to 0.0.0.0 Without Authentication or TLS; OAuth 2.1 + PKCE Correctness; Tool Name Collision / Spoofing; Prompt / Metadata Injection in Tool Descriptions; CORS Misuse: Wildcard with Credentials; Transport Security; Authentication and Authorization Checks; OAuth Proxy Per-Client Consent.

Skills and plugins rules: Capability Transparency; Consent Suppression and Human-in-the-Loop Subversion; Host-Credential and Conversation-Context Harvesting and Exfiltration; Format-Native Out-of-Band Execution and Host Control-Plane Persistence; Over-Declared or Over-Privileged Manifest Scope; Injection and Concealed Directives in Manifests and Instructions; Package Provenance, Integrity, and Post-Approval Mutation; Impersonation, Typosquatting, and Publisher Spoofing.

Source notes for editorial verification

1. Public-repository counts: 75 repositories with completed assessments since July 30, 2026; 119 of 143 public attempts completed. Failed attempts produced no report.

2. Component counts: latest completed three-stage assessment per repository as of October 9, 2026; 1,707 components across 72 repositories, including six components of other types beyond the 150 MCP servers and 1,551 skills.

3. Rule counts: production release branch as of October 9, 2026.

4. Reviewer vendor: 145 of 147 completed assessments used models from different vendors in first pass and review; reviewer receives the first-pass findings.

5. Review changes: 136 assessments created August 25 through October 9, 2026; 21 involved non-public customers. No ground truth is available. Reconciliation accepted 183 of 219 proposed withdrawals and 643 of 645 proposed additions.

6. Largest assessment: maximum number of distinct components scored in one completed production assessment.

7. Context-adjusted severity: findings created since August 25 and completed after September 11, 2026; figures recomputed from stored findings using the deployed function.

8. Benchmark: not included in this paper. Results from BlueRock's 12-flaw test repository came from different runs under different conditions and do not support a detection-rate claim.

9. Rule growth: 22 rules on April 11, 2026; 34 from August 10, 2026.

10. Rule updates: merges on the main branch from July 1 through October 9, 2026; not every update is linked to a specific attack.

11. Repository popularity: median 31,802 stars; stars read October 8, 2026; 74 of 75 repositories had 1,000 or more stars.

12. Component counts by repository: 39 of 75 had at least eight separately scored components; median eight.

13. Tool inventory: counts from each repository's latest completed assessment; 56% of inventoried tools received an explicit second-review verdict.