The cybersecurity industry's testing standard community

Guidelines for Testing of Agentic Security Products

Contributing Members:

- Luis Corrons (Gen Digital)
- Jan Miller (OPSWAT)
- Igor Luckel (AV-TEST)
- Megan Squire (F-Secure)
- Nima Bagheri (Venak Security)
- Jamz Yaneza (Individual Member)
- Erica Marotta (Artifact Security)
- Alexander Vukcevic (Avira)
- Adrian Scibor (AVLab Cybersecurity Foundation)
- Cameron Camp (SecureIQLab)
- Jan Sirmer (Gen Digital)
- Dimitrios Tsarouchas (Artifact Security)
- Savino Dambra (Gen Digital)
- Stefan Dumitrascu (Artifact Security)
- Jakub Křoustek (Gen Digital)
- Working Group admins: John Hawes (AMTSO), Scott Jeffreys (AMTSO)

Version: 1.0

Date: 2026-09-02 (Approved and adopted by AMTSO membership)

Copyright © 2026 Anti-Malware Testing Standards Organization, Inc. All rights reserved.

Notice and Disclaimer of Liability Concerning the Use of AMTSO Documents

This document is published with the understanding that AMTSO members are supplying this information for general educational purposes only. No professional engineering or any other professional services or advice is being offered hereby. Therefore, you must use your own skill and judgment when reviewing this document and not solely rely on the information provided herein.

AMTSO believes that the information in this document is accurate as of the date of publication although it has not verified its accuracy or determined if there are any errors. Further, such information is subject to change without notice and AMTSO is under no obligation to provide any updates or corrections.

You understand and agree that this document is provided to you exclusively on an as-is basis without any representations or warranties of any kind whether express, implied or statutory. Without limiting the foregoing, AMTSO expressly disclaims all warranties of merchantability, non-infringement, continuous operation, completeness, quality, accuracy and fitness for a particular purpose.

In no event shall AMTSO be liable for any damages or losses of any kind (including, without limitation, any lost profits, lost data or business interruption) arising directly or indirectly out of any use of this document including, without limitation, any direct, indirect, special, incidental, consequential, exemplary and punitive damages regardless of whether any person or entity was advised of the possibility of such damages.

This document is protected by AMTSO's intellectual property rights and may be additionally protected by the intellectual property rights of others.

Copyright © 2026 Anti-Malware Testing Standards Organization, Inc. All rights reserved.

AMTSO Guidelines for Testing of Agentic Security Products

Terminology

Term	Meaning
Security product, Product, PUT, DUT	Interchangeable terms for the product under test. The term may refer to an API gateway, in-agent control, browser or endpoint component, OS-level control, cloud service, agent platform feature, or integrated security platform.
Tester	The organization or person conducting the evaluation.
Agentic system	A system in which an AI model can plan, use tools, retrieve data, call APIs, access files, write memory, or otherwise take actions on behalf of a user or another system.
Agent host	The runtime, application, browser assistant, development tool, or platform in which the agent executes.
Tool	A capability the agent can invoke, including a function call, API call, browser action, file operation, database query, extension, skill, plugin, MCP tool, or other external capability.
Tool registration	The point at which a tool definition, schema, description, capability declaration or server endpoint is made available to the agent.
MCP	Model Context Protocol. In this document MCP is used as one example of an agent tool and server ecosystem. The guidelines should also apply to comparable tool protocols or extension models.
Prompt injection	An attempt to manipulate the agent through instructions placed in user input, retrieved content, tool descriptions, memory, or other context consumed by the agent.
Indirect prompt injection	A prompt injection delivered through content the agent retrieves or processes, such as a web page, email, document, API response, repository file, calendar item or ticket.
FP	False Positive. Legitimate agent activity or content incorrectly classified as malicious.
FN	False Negative. A malicious scenario, action or content item missed by the product under test.
Detection	A product action that identifies and records, reports or notifies that a malicious or suspicious event occurred.
Prevention	A product action that stops or meaningfully disrupts the malicious outcome, such as blocking a tool call, suppressing a payload, redacting sensitive data, requiring user approval, or terminating the session.
Model refusal	A case where the AI model itself refuses to perform an unsafe instruction without intervention by the security product. This should be recorded separately from product detection or prevention.

Scope

The scope of this document is to define guidelines for testing security products that claim to protect AI agents or agentic systems from security threats. The document is intended to be product-architecture neutral and should apply to enterprise and consumer environments where an agent can process user requests, consume external content, use tools, or take actions.

In this context, agentic security products are products that do one or more of the following:

1. Inspect or control user prompts, retrieved content, model outputs, tool calls, tool registration, memory writes, agent-to-agent messages, or other parts of an agent interaction lifecycle.
2. Claim to detect, prevent, warn about, constrain, isolate, log, or otherwise reduce security harm caused by attacks against agentic systems.
3. Operate as an independent product, a product feature, an agent-platform control, an API proxy, an in-agent library, an endpoint or browser component, an OS-level control, a cloud service, or a combination of these.

Copyright © 2026 Anti-Malware Testing Standards Organization, Inc. All rights reserved.

The first version of these guidelines focuses on scenarios where the tester can observe and validate the agent action chain. It does not attempt to solve every problem related to LLM security, AI safety, model alignment, model training, infrastructure hardening, or general software security.

Out of scope for the first version

- Evaluation of AI model alignment, helpfulness, truthfulness, or jailbreak resistance as a standalone model property.
- Training-time attacks such as data poisoning, model backdoors, model extraction or model supply-chain attacks where no agent behavior is exercised.
- Traditional endpoint, network, email, web or malware protection testing, except where these controls are part of an agentic protection claim and the agentic chain is being evaluated.
- Availability attacks against AI infrastructure, such as DDoS, cloud resource exhaustion or GPU capacity attacks.
- General compliance audits of AI governance programs.

General principles of testing agentic security products

1. Tests and benchmarks should focus on the additional value delivered to the user, not on the technical implementation chosen by the vendor. A product that protects through an API proxy should not score differently from a product that protects through an in-agent hook, endpoint control or platform-native control if the user protection outcome is equivalent.
2. The tester should evaluate the full attack chain wherever possible. Agentic attacks often look benign at one step and malicious only when prompts, retrieved content, tool calls, memory changes and output are viewed together.
3. A product should receive credit for effective protection at any relevant point in the chain, provided the malicious outcome is prevented or the user is meaningfully protected. For example, blocking a dangerous outbound tool call can be valid even if an earlier prompt injection was not blocked.
4. The tester should distinguish product detection and prevention from model refusal. If the model refuses an unsafe request on its own, that result should be reported, but it should not be counted as product detection or prevention unless the product demonstrably caused the refusal or intervention.
5. Test results should reflect the scope of the product claim. A product that claims to protect browser-based consumer agents should be tested differently from a product that claims to monitor enterprise developer agents, database-connected agents or MCP servers.
6. False positives, usability impact and user experience should be measured. An agentic security product that blocks normal workflows too aggressively may reduce user value, even if attack detection appears strong.
7. Test environments should be safe, controlled and reproducible. No test should create unnecessary risk to public systems, real users, production data or third-party infrastructure. Where test results contain information which could disclose potential novel attack vectors or security bypasses, testers should follow appropriate best practices for responsible disclosure and remediation prior to publication, or redact outputs to prevent unsafe disclosure.
8. The methodology should disclose enough information to allow vendors, testers and readers to understand what was tested, how it was tested, what was not tested, and where the results may not generalize. As test scenarios may be non-deterministic and unreproducible, ample records should be kept of test cases and results, including screenshots/recordings and detailed logging of all activities (including logs from security products), to allow clear understanding of any issues observed and to support remediation.

Copyright © 2026 Anti-Malware Testing Standards Organization, Inc. All rights reserved.

Different attack vectors

In this document, attack vectors are grouped by how malicious influence reaches the agent and where the harmful outcome occurs. These vectors may be used separately or in combination. A single scenario may begin as one vector and be prevented at a different inspection point. Vectors are presented in order of significance/relevance, testers limiting their tests to certain vectors should focus on those listed first.

1. Direct agent input

Direct agent input attacks are initiated through normal user-facing channels. The attacker may be a malicious user, a compromised account, a malicious insider, or a legitimate user who is tricked into submitting attacker-controlled instructions.

Examples include direct prompt injection, authority impersonation, attempts to extract system prompts, attempts to enumerate tools or data sources, requests to override policies, and multi-turn manipulation where each individual message appears harmless but the sequence leads to an unsafe action.

The tester should consider whether the product detects or prevents the malicious intent before model processing, during planning, during tool invocation, or before the final response is shown to the user.

2. Indirect content-mediated attacks

Indirect attacks are delivered through content that the agent retrieves, reads, summarizes or otherwise processes. The user may have requested an apparently legitimate task, while the malicious instruction is hidden or embedded inside the content consumed by the agent.

Examples include poisoned web pages, documents, emails, tickets, calendar invitations, repository files, API responses, search results and retrieved knowledge base entries. Hiding techniques may include HTML comments, off-screen text, invisible text, markdown tricks, metadata, encoded instructions, or socially convincing instructions addressed to the model.

The tester should account for the fact that a product may protect at several places: content retrieval, content normalization, model context construction, planning, tool invocation, output inspection, or user confirmation.

3. Tool, skill, extension and MCP supply chain

Tool and extension supply-chain attacks target the capabilities that an agent can use. The attacker may publish, compromise, modify or impersonate a tool, skill, plugin, extension or MCP server. The malicious behavior may be visible at registration time, execution time, or only after a delayed change.

Examples include malicious tool descriptions, schema changes that introduce dangerous defaults, tool name shadowing across servers, capability expansion after trust has been established, and time-delayed behavior changes. These attacks should not be reduced to prompt injection only; they often combine tool metadata, protocol behavior, trust decisions and action execution.

The tester should record whether the product inspects tool registration, tool descriptions, schemas, server identity, capability changes, tool calls, parameters and outbound network behavior.

4. Outbound action and data exfiltration

Outbound action attacks are those in which the harmful effect happens through an action taken by the agent, rather than through the text of the prompt alone. These may include sending data to an external endpoint, modifying a file, making a purchase, changing an account setting, opening a pull request, sending a message, invoking a shell command, or querying a database outside the original task scope.

A product may validly protect users by blocking or constraining the outbound action even if earlier stages of the attack reached the model. Testers should avoid scoring only the first point of detection and should instead evaluate whether the user was protected from the harmful outcome.

Copyright © 2026 Anti-Malware Testing Standards Organization, Inc. All rights reserved.

5. Memory, context and cross-agent propagation

Some attacks aim to persist beyond a single interaction or propagate between agents. They may write malicious instructions into memory, documents, tickets, shared workspaces or messages that another agent later consumes.

These attacks are difficult to test consistently, but they are relevant to agentic systems because harmful instructions may become latent until a future task or a different agent activates them.

Example diagram of a possible environment

The following example shows a simplified environment for testing agentic security products. The exact architecture may vary, but the tester should document where the product is positioned and which inspection points it can observe.

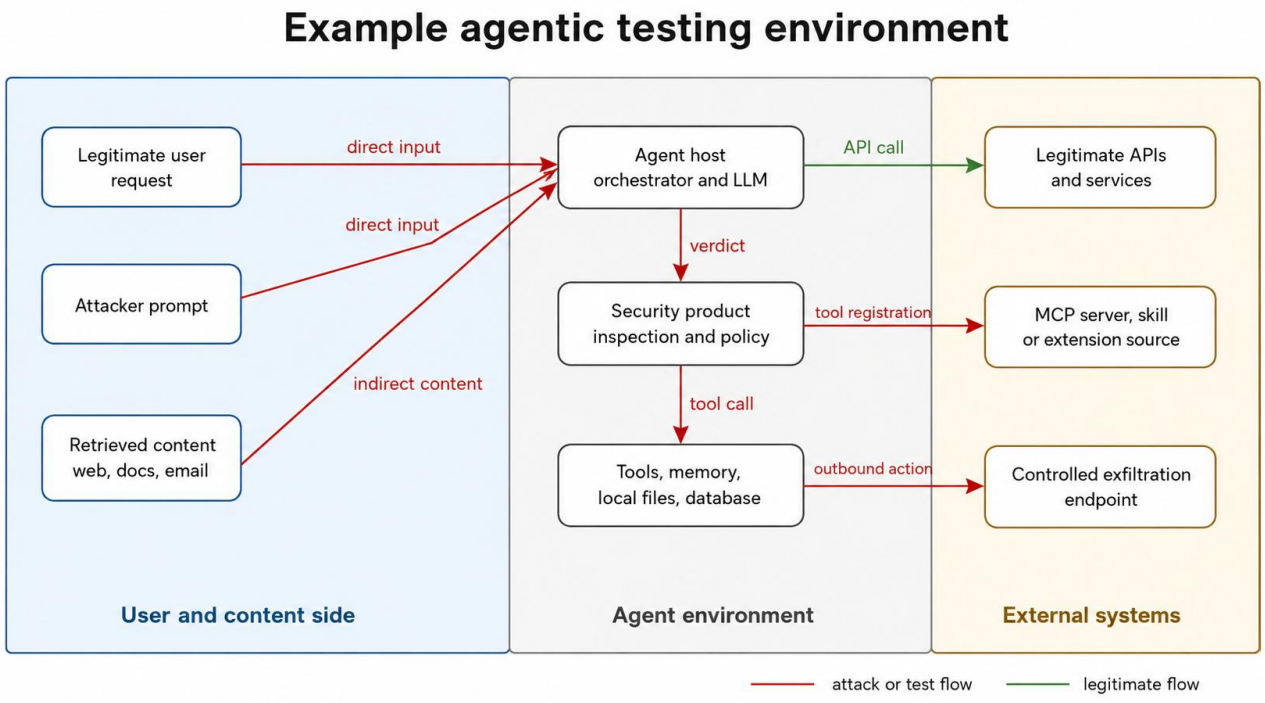

Figure 1. Example environment for evaluating direct input, indirect content, tool registration, tool execution and outbound action controls.

Test case selection

Because the agentic threat landscape is volatile and product architectures differ significantly, test case selection should be driven by the scope of the product under test and by realistic user harm. Testers should avoid building a corpus that overfits to one vendor architecture, one model, one agent host, or one type of prompt injection.

A test case should normally be considered valid only if the malicious outcome can occur in the same environment when the security product is absent, disabled, or configured not to intervene. If an attack payload cannot cause a meaningful unsafe action even without the product, it should not be treated as a missed detection. It may still be useful as a benign or control scenario.

Testers should classify test cases using at least the following dimensions:

- Attack vector: direct input, indirect content, tool or extension supply chain, outbound action, memory or cross-agent propagation.

Copyright © 2026 Anti-Malware Testing Standards Organization, Inc. All rights reserved.

- Target of protection: agent host, tool call, data source, user account, browser session, local file system, enterprise application, consumer account, or external service.
- Environment type: consumer, small business, enterprise, developer, browser-based, endpoint-based, cloud-based, or mixed.
- Harm type: data exposure, credential exposure, unauthorized action, persistence, lateral movement, social engineering amplification, fraud enablement, malware delivery, or policy bypass.
- Severity: low, medium, high or critical, with criteria disclosed by the tester.
- Required capability: retrieval, memory, database access, file access, network access, shell access, identity access, messaging access, payment or transaction capability, or no tool use.

As with malware sample selection in other domains, the source and distribution of test cases should be disclosed at a useful level. If payloads or scenarios are provided by participating vendors, testers should avoid disproportionate weighting from any single vendor and should report the distribution where feasible. Publicly known scenarios, researcher-provided scenarios and tester-created scenarios can all be useful, but their origin and intended purpose should be documented.

Active and inactive scenarios

Agentic scenarios can become inactive in ways that resemble inactive malware samples. For example, an exfiltration endpoint may be unreachable, a tool may no longer register, a vulnerable agent host may have changed behavior, or a model update may prevent the attack before the product under test is involved.

A scenario should be treated as active or valid if the malicious functionality is still carried out, or if the test harness can safely emulate the attacker-controlled infrastructure and verify the unsafe outcome. A scenario should be treated as inactive if the malicious outcome cannot happen and cannot be meaningfully emulated in the controlled environment.

Where feasible, testers should validate scenario activity before and after the test run. Any change in state should be noted in the report.

Determination of detection and prevention

Agentic systems can be difficult to evaluate because the model, the agent host and the security product may all influence the outcome. It is therefore important to state clearly what counts as detection, prevention, refusal and failure.

Apparent and measurable product outcomes may include blocking an input, suppressing retrieved content, refusing tool registration, blocking or modifying a tool call, requiring user confirmation, preventing data exposure, redacting sensitive output, terminating a session, raising an alert, logging an event, or notifying the user or administrator.

The tester should prefer observable evidence over vendor-reported assertions. Observable evidence may include product logs, API verdicts, agent traces, tool call traces, network captures, blocked requests, redacted outputs, user-facing warnings, administrative alerts, or changes in the test harness state.

Outcome	Guideline
Prevented	The product stopped or materially disrupted the malicious outcome. This may happen before model processing, during planning, at tool registration, at tool execution, at output generation, or at user confirmation.
Detected but not prevented	The product identified the event as suspicious or malicious, but the harmful action still completed. This should be counted separately from prevention.
Model refusal	The model refused the request without demonstrable product intervention. This is useful context, but should not be credited as product detection or prevention.
Model recognition	The model mentioned that content or behavior looked suspicious, but no product action occurred and the harmful action was not blocked.

Copyright © 2026 Anti-Malware Testing Standards Organization, Inc. All rights reserved.

Outcome	Guideline
Missed or successful attack	The malicious action completed and no relevant product detection or prevention occurred.
Inconclusive	The tester cannot determine the outcome because of non-determinism, insufficient instrumentation, environmental failure, or contradictory evidence.

When the product under test reports only aggregated detections or does not expose detailed event data, the tester should state the limitation clearly. In such cases, the tester may need to rely on harness-level observation, for example whether a tool call reached the destination, whether data appeared at the controlled exfiltration endpoint, or whether a file, memory entry or external account was modified.

Preparation of the environment

In an ideal test, all scenarios would run in a fully controllable environment using realistic agent hosts, realistic tools, realistic data flows and realistic user workflows. In practice, compromises will often be necessary. Any compromise that may affect the interpretation of results should be disclosed.

The environment should include only synthetic or approved data. It should not use real customer records, real credentials, real payment instruments, live user accounts, or uncontrolled third-party infrastructure. If the test requires external-looking services, they should be hosted in infrastructure controlled by the tester or otherwise isolated from public harm.

The tester should document at least the following:

- Agent host, agent framework and version.
- LLM provider, model name, model version where available, system prompt handling, temperature or sampling settings where configurable, and relevant safety settings.
- Security product name, version, deployment mode, configuration, policy settings and integration point.
- Tool set, tool schemas, permissions, data sources and external endpoints available to the agent.
- Test data set, including the type of synthetic sensitive data used and the expected sensitive fields.
- Network isolation, logging, test harness controls and reset procedures.
- Known limitations of the environment compared with a real deployment.

Baseline validation

Before testing the security product, the tester should validate that the agentic scenario can execute in the intended way without product intervention. This includes both benign workflows and malicious workflows. A benign workflow should complete successfully. A malicious workflow should reach the unsafe outcome if no protection is active, unless the purpose of the test is explicitly to measure product behavior in the presence of model refusal.

A stronger approach is to run baseline validation before and after the protected test run. This helps identify whether model updates, tool changes, endpoint changes or test harness failures changed the validity of the scenario.

State reset and repeatability

Agentic tests are affected by model non-determinism, persistent memory, context windows, cached tool results, user approval state and learned product state. The tester should define what state is reset between runs and what state is intentionally preserved.

For independent scenarios, the agent conversation state, short-term memory, local files, test database, external endpoint state and tool server state should normally be reset. For scenarios that explicitly test persistence, memory poisoning, delayed tool changes or learned baselines, the preserved state should be described.

Copyright © 2026 Anti-Malware Testing Standards Organization, Inc. All rights reserved.

Where model non-determinism materially affects the result, the tester should execute repeated runs and report the distribution of outcomes rather than hiding variability behind a single pass or fail result. The exact number of repetitions should be set by the methodology and may vary by scenario complexity. Testers should ensure their policy on handling non-determinism is clearly explained and justified, including where possible details of external inputs outside the tester's control (eg web resources accessed by agents during the course of testing), and statistical data showing the confidence, consistency, and error-margin of the results.

Agent or system under attack

If the tester does not use a real deployed agent environment, the replacement should mimic the intended real environment as closely as possible. For example, a developer-agent test should include repository files, build scripts, local secrets placeholders, tool permissions and code-editing workflows. A browser-agent test should include realistic pages, forms, session state and navigation paths. An enterprise assistant test should include realistic data sources, access permissions and business workflows.

Synthetic agents can be useful, but testers should reflect their use in the report. The report should explain which behaviors were real, which were simulated, which tool calls were stubbed, and which external services were controlled by the tester.

Advanced threat and supply-chain emulation

Emulating advanced agentic attacks can be challenging. When a vendor claims protection against advanced tool abuse, delayed MCP attacks, complex prompt injection, cross-agent compromise, data exfiltration or malicious autonomous behavior, a sufficiently skilled and resourced tester may validate those claims using controlled infrastructure.

The tester should avoid creating unnecessary risk. Malicious tool servers, exfiltration endpoints, poisoned documents, fake APIs, credential stores, code repositories and web pages should be locally hosted, sinkholed, access-controlled, or otherwise prevented from affecting third parties. Payloads should use synthetic secrets and markers rather than real credentials.

Advanced scenarios may include delayed tool behavior changes, malicious tool descriptions, schema changes with dangerous defaults, tool name shadowing, chained tool abuse, hidden instructions in retrieved content, data staging followed by exfiltration, or persistence through memory. The report should identify which of these were tested and which were not.

Testing of specific security functionality

Disclaimer

Because agentic attacks often involve multiple stages, it is possible to test each stage individually or as part of a complete end-to-end scenario. Whole-product testing should include a balance of staged tests and end-to-end tests. The methodology documentation should clearly state when a result represents stage-level protection and when it represents full attack-chain protection.

Reconnaissance visibility

Many agentic attacks begin with reconnaissance. The attacker may try to identify available tools, data sources, memory behavior, system prompt boundaries, access permissions, model identity, guardrail sensitivity or rate limit behavior. These actions may look similar to legitimate troubleshooting or power-user behavior.

Tests for reconnaissance visibility should determine whether the product can identify suspicious enumeration without blocking ordinary user activity. Examples include repeated attempts to list tools, infer database schemas, extract hidden instructions, probe policy boundaries, or discover accessible file paths and credentials.

Copyright © 2026 Anti-Malware Testing Standards Organization, Inc. All rights reserved.

The tester should be cautious in scoring reconnaissance. Some products may intentionally warn, log or apply risk scoring at this stage rather than block the request. That may be an appropriate user-protection strategy if later harmful actions are prevented.

Direct prompt injection and tool misuse

Direct prompt injection tests should evaluate user-submitted instructions that attempt to override policy, impersonate authority, redirect the agent goal, extract hidden instructions, misuse available tools, or access data beyond the user request.

A useful scenario should define the legitimate user objective, the malicious instruction, the available tools, the expected unsafe outcome, and the point at which the product could reasonably intervene. Multi-turn tests should include the full sequence and should not be scored as isolated single messages unless the methodology explicitly states this limitation.

Indirect prompt injection

Indirect prompt injection tests should evaluate malicious instructions embedded in content retrieved or processed by the agent. The content should be realistic for the tested environment, such as a web page for browser agents, an issue or repository file for developer agents, a document for productivity agents, or an email or ticket for enterprise assistants.

The tester should record whether the product inspected the retrieved content, the constructed model context, the model output, the tool plan, the tool call, or the final user-facing response. If the product only detects the attack after sensitive data has already been sent to an external endpoint, this should not be counted as prevention.

Tool, extension and MCP registration

Tool and extension registration tests should evaluate whether the product can detect malicious or suspicious capabilities before execution, where the product claims to support such inspection. This may include tool names, descriptions, schemas, permission requests, server identity, capability changes and dangerous defaults.

Delayed behavior should be tested carefully. The security product may need to retain state from an earlier benign registration phase to detect a later malicious pivot. If the tester resets the product state between phases, the report should state that the scenario did not test learned baseline behavior.

Outbound action and data exposure prevention

Data exposure and unsafe action tests should evaluate whether the product prevents the agent from sending sensitive data, credentials, internal records or unauthorized requests to destinations outside the approved task scope. The tester should use controlled endpoints that confirm whether the harmful action occurred.

Examples include exporting database rows to a webhook, including secrets in a support ticket, sending confidential document content to a fake external service, writing a malicious file, executing a shell command, modifying a repository, or submitting a form that changes an account setting.

Memory and persistence

Memory and persistence tests should evaluate whether malicious instructions can be written into persistent memory, files, shared workspaces, tickets or other artifacts that the agent later consumes. The tester should confirm both the write stage and the later activation stage.

These tests are likely to be environment-specific and may be more suitable for supplemental testing until the working group agrees a common baseline.

Copyright © 2026 Anti-Malware Testing Standards Organization, Inc. All rights reserved.

Cross-agent scenarios

Cross-agent testing should evaluate whether a malicious instruction or artifact created by one agent can affect another agent with different privileges or tool access. This is relevant for orchestrator-worker systems, shared workspaces and enterprise workflows where agents exchange messages or artifacts.

For a first version, cross-agent tests should be limited to simple and reproducible scenarios with two agents, defined permissions and a controlled communication channel. More complex multi-agent topologies should be considered future work.

Multimodal scenarios

Multimodal attacks may place instructions in images, screenshots, audio transcripts, documents or mixed media. Products that claim multimodal inspection should be tested with relevant content. Products that do not process multimodal input should not be penalized for not detecting attacks outside their claimed scope.

The tester should report whether the content was processed by OCR, vision models, document parsers, metadata extraction or another mechanism, because these choices may materially change the test result.

False positive testing

False positive testing should be performed alongside attack testing. It should include legitimate workflows that resemble risky behavior but are authorized and expected in context.

Examples include legitimate database queries by an analyst, authorized bulk export to an approved internal destination, normal use of developer tools, benign web summarization containing security terminology, safe credential scanning requested by a developer, normal MCP tool registration from a trusted server, and legitimate administrative automation.

False positive results should not be hidden in aggregate attack scores. They should be reported separately and, where useful, broken down by workflow type and severity of user impact.

Benchmarking of performance and user impact

Agentic security products may add latency, require user approvals, block workflows, change model context, redact output, or alter tool behavior. Performance and user impact should therefore be considered, especially for products deployed inline with user workflows.

Performance benchmarking is a complex topic and may be outside the scope of the first guideline version. However, testers should consider the following points:

Operation differentiation

Latency should be measured in relation to the operation being protected. A verdict on a tool registration event may have different latency expectations from a verdict on a chat prompt, browser action, shell command, database query, file write or outbound API call. The customer use case also matters; a consumer browser assistant and an enterprise automation agent may have different tolerance for delay.

Baseline

It can be informative to measure the agent workflow with the product disabled, with the product enabled in monitoring-only mode, and with blocking or user-confirmation controls enabled. The baseline should represent the workflow the product would normally replace or augment.

User friction

Some controls prevent harm by requiring user approval, administrator approval or step-up authentication. These outcomes should not be treated as simple latency only. The report should distinguish automated blocking from user-mediated control and should describe whether the approval prompt provides enough information for a reasonable user to make an informed decision.

Copyright © 2026 Anti-Malware Testing Standards Organization, Inc. All rights reserved.

Reporting and transparency

Reports should provide enough detail for readers to understand the test scope, reproduce the environment where appropriate, and interpret the limits of the result. At minimum, reports should include:

- Product name, version, deployment mode, configuration and claimed protection scope.
- Agent host, model, model version where available, tool configuration and data sources.
- Description of each scenario category and the number of scenarios in each category.
- How scenarios were selected, including whether they came from the tester, public research, participating vendors or other sources.
- Definitions of detection, prevention, refusal, failure and inconclusive outcomes used in the test.
- Per-category detection and prevention results, with false positives reported separately.
- Known cases where the model refused without product intervention.
- Observed user impact, including latency, approval prompts, blocked legitimate workflows and other usability effects where measured.
- Any deviations from the published methodology.
- Known limitations of the environment, instrumentation or product integration.

Raw prompts, retrieved content, tool definitions, tool traces, logs and harness outputs should be retained long enough to support vendor review, dispute handling and independent verification where appropriate. Sensitive information and exploit-enabling details should be handled responsibly, especially if a report is intended for public release.

Limitations and future work

The working group expects this area to change quickly. The first guideline aims to be practical and limited rather than trying to solve every agentic security testing problem at once.

- Model dependency: Agent behavior varies across models, versions and safety settings. Results may not transfer across model providers or releases.
- Agent host dependency: The same model may behave differently depending on the host, tool framework, permissions, retrieval layer and memory implementation.
- Non-determinism: Repeated runs may produce different outcomes. The methodology should report variability where it matters.
- Scenario maturity: Some attack classes, especially cross-agent propagation, autonomous long-horizon behavior and complex tool supply-chain compromise, need more pilot testing before stable scoring rules are defined.
- Severity weighting: The first version may report scenarios equally, but future versions should define severity and user harm weighting.
- Benchmark data sets: The working group should consider whether standardized payloads, synthetic data sets and reference environments can be published safely.
- Regulatory alignment: Future versions may map these guidelines to relevant AI security and governance frameworks, but regulatory compliance should not replace empirical protection testing.

Copyright © 2026 Anti-Malware Testing Standards Organization, Inc. All rights reserved.