Penetration Testing and Security Protection of Artificial Intelligence Systems from a Data Security Perspective

Xuanlin Wen

NCB Beijing, Beijing, China
Email: 304489063@qq.com
Manuscript received August 29, 2024; accepted October 20, 2024; published December 29, 2024.

***Abstract*—Artificial Intelligence (AI) systems continuously aggregate high-value data throughout the entire processes of training, inference, and operation, evolving from technical components into concentrated carriers of data assets, and accordingly have become high-priority targets for attackers [1, 2]. Taking data security as the analytical thread, and drawing on the author's nearly two years of professional experience, this paper discusses the application of traditional penetration testing frameworks and security protection frameworks to the cyber offense and defense of AI systems. Regarding penetration testing, in view of the complex technology stack and dispersed attack surface of AI systems, the systems are first decoupled hierarchically; with reference to the cloud service model, they are divided into an infrastructure layer, a platform component layer, and a model application layer. On this basis, the attack-phase division and the tactic-technique mapping provided by Kill Chain and ATT&CK remain methodologically applicable [3, 4]. Combining them with the three-layer asset morphology allows the construction of an integrated path of risk analysis and test verification, forming an important data acquisition chain that runs from the server side through to the client side. Regarding security protection, following the strategy of "macro emphasis on management, micro emphasis on technology," the macro level reduces the number of strategy enforcement points, improves their efficiency, and lowers the complexity of their coordination by compressing the attack surface and integrating information flows. The micro level emphasizes technical requirements in the concrete implementation of strategy enforcement points, referring to models such as IPDRR and OODA [5] to carry out identification, protection, detection, response, recovery, and dynamic cycles. Section IV presents two sets of controlled experiments implemented and executed in a locally simulated range, validating the above methodology.**

***Keywords*—data security, artificial intelligence, penetration testing, Kill Chain, ATT&CK, IPDRR, OODA, attack surface reduction**

I. Introduction

The value of data security within cybersecurity is becoming more prominent than ever before. As data becomes a core factor of production, the focus of cyber offense and defense has shifted from system control to the protection and theft of data [1, 2].

AI systems amplify this trend further: their data has dispersed out of databases and file servers, being split into training corpora, fine-tuning samples, vector embeddings, inference contexts, and session logs, scattered across data pipelines, feature stores, model weights, vector databases, and even terminal applications.

This state of "data dispersal" is simultaneously complicated along three dimensions: form, ownership, and lifecycle.

The same body of knowledge is repeatedly copied and derived in multiple forms—original documents, vector slices, fine-tuning parameters, and context fragments; data flows among data engineers, model developers, platform operators, and end users, each with a different permission model; training data can be "memorized" by weights, and inference data can be retained by sessions and caches, making deletion and traceability far more difficult than in traditional databases.

The direct consequence is the failure of database-centered security boundaries: every transfer of data constitutes a new point of exposure, while value density is rising.

If data theft is taken as the objective, the attack surface of an AI system is markedly broader than that of a business system of comparable scale: at the code level there are vulnerabilities; at the data level there are poisoning risks; at the model level there are adversarial examples and prompt injection risks; and at the supply chain level there are untrustworthiness risks of pre-trained models, third-party plugins, and open-source frameworks [6, 7].

The success criterion for penetration testing and security protection accordingly shifts from "whether the system is controlled" to "whether the data is touched".

The shift in criterion has not shaken the foundations of methodology. The situation of AI systems in cybersecurity is similar to that when early cloud platforms and big data platforms first emerged: they overlay a new technology stack upon the original one, bringing new asset forms and attack surfaces; the basic behavioral logic of attackers still continues, and the basic order of defense still holds.

Attackers must still conduct reconnaissance first, then construct payloads, deliver them, exploit them, and exfiltrate data, while defenders must still complete identification, protection, detection, response, and recovery. Therefore, the overall approach to AI-oriented penetration and protection can still follow classical frameworks such as Kill Chain, ATT&CK, IPDRR, and OODA [3–5].

What needs to be done is to expand the asset inventory and technical entries within these frameworks.

This raises two questions: whether traditional penetration testing strategies and steps remain effective in AI scenarios and how they can be implemented; and what division of labor the resource allocation of security protection should follow and what its concrete handles are.

Section II first decouples AI systems hierarchically, then

argues the logic of continuing to use Kill Chain and ATT&CK, and provides testing approaches; Section III responds to the division of labor between macro-level management and micro-level technology and to concrete handles. Section IV conducts two sets of controlled experiments in a locally simulated range to verify the effectiveness of the aforementioned frameworks.

II. Application of Traditional Attack Frameworks in the Penetration Testing of Artificial Intelligence Systems

This part first decouples AI systems hierarchically, then argues the logic of continuing to use Kill Chain and ATT&CK in AI scenarios, and finally organizes penetration testing along the three layers.

A. Hierarchical Decoupling as a Prerequisite for Addressing System Complexity

Discussing the penetration testing of AI systems, one first confronts a practical difficulty: the systems are so complex that there is no place to begin. This complexity stems from three aspects.

The first is the stacking of technology stacks: AI systems are built upon networks, servers, and cloud platforms, upon which data pipelines, training frameworks, model repositories, inference services, vector databases, agent orchestration, and front-end applications are overlaid as multiple layers of components, each with independent configurations, permissions, and vulnerability surfaces.

The second is the lengthening of data chains: the same piece of data runs from collection, cleaning, training, and fine-tuning, through indexing, to inference, sessions, and terminal retention, crossing multiple components and multiple responsible parties, so security boundaries are no longer concentrated at a single point.

The third is the addition of new asset forms: model weights, vector embeddings, prompt templates, tool invocation configurations, and other new asset forms belong neither to traditional code nor are they fully equivalent to traditional data, and their vulnerabilities and leakage consequences still lack mature analytical frameworks.

If the system is assessed as a whole, conclusions often remain at the general judgment that "the attack surface is broad and the risk is high," which makes it difficult both to locate concrete problems and to design targeted test cases.

A feasible approach is hierarchical decoupling: according to ownership of control and level of abstraction, the system is decomposed into several relatively independent layers, the technology stack, asset forms, and trust boundaries of each layer are refined, the data leakage paths of each layer are then analyzed one by one, and finally the conclusions of the layers are chained together to reconstruct the complete picture of data flows.

The layered approach draws on the mature practice of IaaS, PaaS, and SaaS in the cloud service domain: the infrastructure layer corresponds to IaaS, covering networks, servers, cloud platforms, and container orchestration systems, with asset forms being computing power, storage, and network resources as well as the runtime environments deployed upon them.

Risks focus on underlying configuration defects, privilege transitions, and transmission exposure, and data leakage is characterized by large volume, broad scope, and bypassing of business logic.

The platform component layer roughly corresponds to PaaS and extends toward the model side, including databases, model repositories, vector databases, and training scheduling platforms, with asset forms being structured data, model weights, fine-tuning samples, vector slices, and knowledge base documents; risks focus on access control failures, missing tenant isolation, and implicit trust between components, and data leakage is characterized by bulk extraction and lateral diffusion.

The model application layer corresponds to SaaS and extends to the client side, covering inference interfaces, agent orchestration, front-end applications, and terminal caches, with asset forms being inference contexts, session histories, tool invocation parameters, and terminal retained data; risks focus on prompt injection, unauthorized tool invocation, and session data residue, and data leakage is characterized by single-point exfiltration that is difficult to recover.

B. Kill Chain and ATT&CK Remain Applicable in AI Scenarios

On the basis of layering, we next examine whether the methodology is applicable. Kill Chain decomposes attacks into seven stages—reconnaissance, weaponization, delivery, exploitation, installation, command and control, and actions on objectives—enabling defenders to set barriers at any link [3]; ATT&CK further refines each stage into observable tactics and techniques, forming a shared language for offense and defense [4].

The two describe the behavioral logic of attackers and can be understood in isolation from any specific technology stack. Although AI systems introduce two new categories of assets—models and data—attackers must still first conduct reconnaissance, then construct payloads, then deliver them, then exploit them, and finally exfiltrate data; this behavioral sequence has not changed.

Therefore, the continuation at the methodological level holds; what needs to be adjusted is the "asset inventory" and "technical entries," not the framework itself. The ATLAS knowledge base proposed by MITRE is essentially an expansion of ATT&CK entries for the AI domain [8], and its very existence confirms the feasibility of the "continue and extend" path.

C. Organizing Penetration Testing Along the Three Layers

Along the aforementioned three-layer division, risk analysis and penetration testing are organized according to the same set of three layers: first identify the leakage paths of each layer, then verify their exploitability with the approach of "probing assets—verifying exploitation—confirming data touch", so that every risk has a corresponding verification direction and every test finding can be attributed to a definite risk.

The testing approaches for each layer are elaborated below, with specific tools and commands omitted.

The infrastructure layer is the network and computing foundation on which AI systems run, and its risks mainly

stem from configuration defects and unpatched vulnerabilities.

Remote execution, missing permission validation, and other vulnerabilities exposed on boundary devices, gateways, or servers can be exploited to achieve privilege escalation, raising ordinary service accounts step by step to host machines and even cloud platform administrative privileges, thereby bypassing all application-layer protections and directly reading plaintext data in object storage buckets, database instances, and backup files.

Once security group rules and bucket policies are misconfigured, data is exposed to the public internet.

When internal east-west traffic, training data transmission, and model service invocations are unencrypted or transport-layer validation is lax, they can be intercepted by man-in-the-middle attacks, capturing training samples, inference requests, and credentials.

Leakage in this layer is large in volume and broad in scope, and because it bypasses business logic it is difficult for application-side auditing to discover.

The testing approach is: first conduct asset mapping to ascertain the exposure of management components such as inference services, various training management planes, interactive development environments, experiment tracking panels, and container orchestration interfaces, as well as cloud storage, and compare them against known vulnerabilities.

Then check container runtime modes, mounts, and the actual permissions of service accounts along the privilege escalation path, assessing the possibility of privilege transition from service accounts to host machines and further to cloud platforms.

On the cloud platform side, verify the anonymous access risks of instance metadata interfaces and object storage buckets.

And verify through packet capture and proxies whether internal traffic is plaintext and whether transport-layer validation is strict.

The platform component layer is the direct convergence point of data, and its risks concentrate on the access control and data governance of components.

Injection vulnerabilities, weak passwords, and permission proliferation of database components can cause structured data to be extracted table by table.

If the model repository, training task scheduling, and inference engine of the model platform have unauthorized access, attackers can not only download model weights but also leverage training task injection to read training corpora and fine-tuning samples.

Vector databases and knowledge bases, due to missing tenant isolation and retrieval permission filtering, allow attackers to construct similarity queries for cross-tenant unauthorized retrieval, "legitimately" extracting others' knowledge base contents in bulk.

Components are often interconnected with implicit trust and long-term credential reuse, so a single compromise spreads laterally along the data pipeline.

The testing approach is: for databases, check weak passwords, injection points, and whether account permissions are excessive.

For model platforms, verify unauthorized access to admin consoles and model repositories, whether model weights can be downloaded anonymously, and whether training tasks can be leveraged to read environment variables and data source connection strings.

For vector databases and knowledge bases, focus on verifying tenant isolation and retrieval permission filtering—construct similarity queries pointing at other tenants' contents with the identity of the local tenant, and if others' slices can be returned, the absence of filtering is confirmed.

At the same time, check key governance by searching for hardcoded keys in configurations, environment variables, version histories, and front-end scripts and verifying their usability.

The risks of the model application layer are the closest to business and people. On the application side, direct and indirect prompt injection can induce the model to divulge system prompts and sensitive information in the context, or to invoke tools without authorization to exfiltrate internal data [9]; if session histories and logs are stored and archived without desensitization, they become leakage sources that can be secondary exploited.

On the terminal side, the local caches, clipboards, screenshots, and export functions of desktop and mobile clients hold complete session data, which can be extracted in bulk once the device is compromised by malware or physical contact.

If the link between users and model services is hijacked by a man-in-the-middle attack, personal information and business secrets are leaked the moment they are submitted.

Once a client holding a legitimate session token is controlled, attackers can also impersonate the user to complete unauthorized export.

Testing revolves around business logic and data residue: on the application side, conduct unauthorized access and parameter injection tests against inference interfaces, check the tool invocation whitelist and plugin loading validation of orchestration frameworks, cover three categories of use cases in prompt injection testing—direct injection, indirect injection, and unauthorized tool invocation—and conduct horizontal privilege escalation tests on export and download interfaces.

On the terminal side, on authorized devices check the on-disk locations and encryption status of local caches and session files, investigate hardcoded keys in mobile apps and browser plugins, test the control of clipboard, screenshot, and offline export channels, and, where conditions permit, verify through proxies whether transport-layer certificate validation is strict.

The risks of the three layers are not isolated.

Chaining the tests together reconstructs a representative attack chain: first, in the infrastructure layer, obtain the inference service API documentation and version information through network mapping, discover unauthorized reading through anonymous access tests against object storage buckets, and directly obtain the original documents of the knowledge base.

In the platform component layer, discover unauthorized access to the model platform admin console, reach the vector database through internal interfaces, construct a cross-tenant similarity query with the local tenant's identity,

confirm the absence of filtering, and read slices without authorization.

Finally, in the model application layer, export others' sessions through a horizontal privilege escalation interface, and obtain complete records and export files in the client cache directory.

D. Attention to Dynamic Processes

Traditional penetration testing frameworks mainly target static test objects.

Since large volumes of data flows in AI systems extend into frequent internal inference and internal-external interaction processes, it is necessary in thinking to expand the test object from "systems" to "model behavior": prompt injection, data poisoning, and model theft should be incorporated into the standard use case library [6, 9].

Output stability and boundary consistency testing should be added.

And the data dimension of penetration test reports should be refined, specifying "which data, what magnitude, and over what duration" has been touched, to replace vague high-risk and medium-risk conclusions.

In addition, the requirements for classification and grading, data minimization, and traceability established by the Data Security Law, the Personal Information Protection Law, and the Interim Measures for the Administration of Generative AI Services [10–13] determine that the ultimate question test conclusions must answer is one of compliance: where data leakage risks lie, what their magnitude is, and how they can be closed in a loop.

This also sets the target for the protection system in the next part.

III. Application of Traditional Protection Frameworks in the Security Monitoring and Protection of Artificial Intelligence Systems

This part responds to the division of labor between macro-level management and micro-level technology in security protection, and to the concrete handles for implementing each.

A. Clarifying the Protection Approach: Top-level Design Before Local Configuration

Just as attack objects require hierarchical decoupling, security protection likewise requires layering. The purpose of protection layering is to clarify the protection approach: keep the big picture in mind and start with the details, completing top-level design first before moving to local policy configuration.

Without layering, protection easily falls into the stacking of local rules and devices from the outset—seemingly defended everywhere, yet in fact directionless with dispersed resources; after layering, one first thinks clearly about where the overall protection skeleton and key checkpoints lie, then implements each checkpoint technically, so that local configuration has a basis and can interlock with one another.

Scientific top-level design is, for AI information systems as for any other large complex structured information system, an important measure that both improves system execution efficiency and fundamentally affects system security—in plain terms, "macro emphasis on management, micro emphasis on technology".

The two core tasks of macro-level management—compressing the attack surface and integrating information flows—are precisely the concrete content of top-level design. Together they serve the optimization of strategy enforcement points, reflected in three objectives: reducing the number of strategy enforcement points, improving their efficiency, and lowering the complexity of coordination among them.

Once top-level design is complete, micro-level technology is responsible for the implementation and operation of the strategy enforcement points themselves, landing the results of macro-level optimization as executable checkpoints; this step is local policy configuration.

B. Macro Level: Compressing the Attack Surface and Integrating Information Flows to Optimize Strategy Enforcement Points

Compressing the attack surface reduces the number of strategy enforcement points from the entry side. Attack surface reduction is the highest-return work at the macro-management level.

Concrete actions include: comprehensively inventorying AI assets, bringing experimental deployments, departmentally self-built applications, and externally invoked services by individuals into the inventory, eliminating shadow systems so that the objects requiring enforcement points are completely enumerable.

Exposing only necessary interfaces externally, physically or logically isolating admin backends from inference services, prohibiting direct public network connection, and contracting the boundary to a few controllable entrances.

Disabling debug endpoints, sample pages, and automatically generated API documentation.

Implementing whitelist downloads for model weights and datasets.

And routing APIs uniformly through gateways so that authentication, rate limiting, and auditing are completed at a unified entrance.

The fewer the entrances, the fewer the locations where enforcement points must be deployed, the more finely rules can be written at each point, and the higher the accuracy of detection and interception; conversely, if the attack surface is uncontrolled, enforcement points can only be sparsely deployed across a large number of low-value entrances, with rule conflicts and missed detections increasing and single-point efficiency being systematically lowered.

The essence of compressing the attack surface is to concentrate limited enforcement capability from "setting points everywhere" to "a few mandatory entrances".

Integrating information flows reduces the number of enforcement points and lowers coordination complexity from the path side. The bypass and fragmentation of information flows are both a management problem and a technical problem; their consequence is that enforcement points have nowhere to be set, or are set in vain, and that points cannot easily coordinate with one another.

The same piece of data is stored in the training pipeline,

vector database, cache, logs, and client, each with a different permission model, and audit logs are mutually disconnected, so managers can neither see clearly where data comes from and where it goes, nor judge which segment is the mandatory path.

The direction of integration is to establish a single trusted data channel: classification and grading results serve as the sole label running through the entire chain, identity authentication is unified into a single system, permission decisions are concentrated in a few decision points, and logs and auditing converge on a unified platform.

Integration brings threefold benefits: first, data flows can be completely characterized, so enforcement points only need to be deployed at positions through which all sensitive data must pass, bypass and replica paths are eliminated, and the number of points is thereby reduced.

Second, all enforcement points share the same set of classification and grading labels and identity systems, so judgment criteria are consistent and single-point efficiency is improved.

Third, logs and alerts converge on a unified platform, so points need not each maintain their own context and coordinate separately, and the complexity of cross-point coordination drops significantly.

Conversely, the more dispersed the information flows, the more enforcement points there are and the more inconsistent the standards, the longer the coordination chain, so that any alert may be confirmed back and forth among multiple systems and the response window is wasted.

The two tasks together answer the same question: how to stop the most data outflows with the fewest enforcement points and the simplest coordination.

Compressing the attack surface reduces the number of points through "entrance convergence," while integrating information flows reduces the number of points and lowers coordination complexity through "path convergence"; together they improve both the working efficiency and the management efficiency of strategy enforcement points, embodying the scientific configuration of strategy enforcement points.

C. *Micro Level: The Technical Implementation of Strategy Enforcement Points Can Refer to Traditional Frameworks such as IPDRR and OODA*

Once top-level design has determined the positions and number of strategy enforcement points, micro-level technology is responsible for setting the points properly and making them effective; this is the core of local policy configuration.

Implementation at this level can refer to frameworks such as IPDRR and OODA, so that these capabilities operate at a pace no slower than the attacker's—and, after computation, potentially faster than the attacker's—landing on every strategy enforcement point.

The deployment positions of strategy enforcement points should cover the mandatory passages of data flows: the unified API gateway undertakes authentication, rate limiting, and content filtering.

The model service front proxy completes compliance checks of prompts and outputs as well as the desensitization of sensitive information.

Data egress checkpoints perform approval and interception of bulk exports and external invocations.

The plugin and tool invocation proxy enforces whitelists and parameter validation.

And the vector retrieval layer implements tenant and classification-level filtering.

Position selection follows one principle: deploy where all sensitive data must pass; duplicating construction on every branch path is unnecessary.

The capability configuration of strategy enforcement points is implemented item by item according to IPDRR.

For identification, clarify the data scope, classification level, and responsible person controlled by each point, and incorporate them into the data asset map.

For protection, implement default denial and data minimization, unify identity authentication, enforce multi-factor authentication and approval for high-privilege operations, complete desensitization or replacement of sensitive fields before they enter the model, and apply watermarking and fingerprinting to high-value data for traceability.

For detection, establish baselines for invocation behavior and content—the former concerning frequency, time periods, parameter distributions, and bulk characteristics, the latter concerning sensitive information patterns and anomalous instruction structures in outputs.

For response, preset playbooks for handling prompt injection, bulk data extraction, and anomalous exfiltration, with one-click blocking, isolation, credential revocation, and forensics capabilities.

For recovery, be able to roll back contaminated data, configurations, and model versions, and re-check whether the enforcement points themselves have been bypassed.

The key to capability configuration lies in the quality of baselines and the interpretability of alerts; the number of rules requires dynamic correction, as in practice frequent false alarms will gradually lead operations staff to ignore alerts, rendering checkpoints nominal.

The operating rhythm of strategy enforcement points is organized according to OODA.

The observation phase converges the logs and signals of each enforcement point on a unified platform, avoiding each point watching on its own.

The judgment phase relies on the same set of classification and grading labels and threat baselines to distinguish normal business invocations from unauthorized extraction, and model hallucination from prompt injection, avoiding misjudgment.

The decision phase selects the intensity of handling according to graded playbooks, balancing risk blocking against harm to normal business.

The action phase feeds handling results back into observation, making the next cycle faster.

The response window of AI systems is shorter than that of traditional systems, as data may already have been taken out the moment it enters the context or lands on the terminal; therefore, the observation and decision of enforcement points must precede data outflow.

The supporting management actions are prerequisites for strategy enforcement points to remain effective: clarify system security owners and data owners, and establish

mandatory gating of pre-launch security assessment and penetration testing.

Establish supply chain admission mechanisms for models and plugins.

Formulate prompt and knowledge base authoring standards, prohibiting the embedding of credentials in prompts.

Conduct regularized red-blue team exercises and emergency drills.

And incorporate data touch into security performance assessment.

These actions produce no direct blocking capability, yet they determine whether strategy enforcement points can be used correctly.

IV. Experimental Testing

This part presents the experimental design and measurement methods, the construction and configuration of the range, the experimental results and analysis, and the conclusions and limitations.

A. Experimental Design and Measurement Methods

To verify the methodology of Sections 2 and 3, this paper designs and implements a locally simulated range, organizing two sets of controlled experiments around the two arguments.

Experiment 1 corresponds to Argument 1, examining whether the phase division of Kill Chain and ATT&CK can directly organize the penetration testing of AI systems: in the same range, with the same attacker, and under the same baseline protection state, tests are executed by two methods—vulnerability scan-driven and Kill Chain-driven—comparing the number of canaries touched, the completeness of attack phases, and the integrity of the data acquisition chain.

Experiment 2 corresponds to Argument 2, examining the protection division of labor of "macro emphasis on management, micro emphasis on technology": the same attacker repeatedly executes under three protection states—the baseline state retains the range's default configuration

The managed state applies macro-management measures (privatizing bucket policies, adding tokens to metadata interfaces, isolating admin consoles, disabling debug endpoints, etc.).

The technical state, on the basis of the managed state, further superimposes micro-technical measures (unified gateway authentication, prompt filtering, retrieval-layer tenant filtering, client data loss prevention, etc.)—comparing metrics such as canary touches, attack surface reduction, alerts and false positives, and detection and response times.

The experiments use the canary touch rate as an operationalized measure of "data touch depth". Canaries are uniquely identified pieces of fake sensitive data embedded in the services of each layer, covering typical asset forms such as cloud access keys, database records, model weights, and session caches; once an attacker reads away a canary, its content appears in the attack responses, leaving evidence.

Thereby, "which data is touched and how much" is transformed from a qualitative judgment into a countable metric, and the evaluation focus shifts from the number of vulnerabilities to the depth of data touch, consistent with the advocacy in Section II regarding refining the data dimension of test reports.

B. Range Construction and Configuration

The range is built according to the three-layer architecture of Section II, consisting of six local services, with the attack engines and protection switches independent of the range services, ensuring that the two sets of experiments repeatedly execute on the same test target.

The infrastructure layer deploys object storage services and cloud metadata and container orchestration; the platform component layer deploys the model platform and vector database and SQL database; the model application layer deploys inference, Retrieval-Augmented Generation (RAG), and agent services, as well as the client. Each layer is preconfigured with vulnerabilities according to the ATT&CK tactic list [4, 8]; the specific configuration is shown in Table 1.

Table 1. Range three-layer services and preconfigured vulnerability configuration

Layer	Service (Port)	Preconfigured Vulnerability (Corresponding ATT&CK Tactic)
Infrastructure layer	Object storage (9101); cloud metadata and container orchestration (9102)	Anonymous-readable bucket policy (Collection); metadata interface without token (Credential Access); unauthorized orchestration interface (Lateral Movement); training task parameter injection (Execution)
Platform component layer	Model platform (9103); vector database and SQL database (9104)	Unauthorized admin console (Initial Access); anonymous model weight download (Collection); SQL injection for table dumping (Credential Access); cross-tenant retrieval without filtering (Lateral Movement)
Model application layer	Inference, RAG and agent (9105); client (9106)	Direct and indirect prompt injection (Defense Evasion, Execution); unauthorized tool invocation (Execution); horizontal privilege escalation export (Collection); hardcoded key at debug endpoint (Credential Access)

Canaries are embedded at 14 locations across the three layers; the inventory is shown in Table 2. All data are self-constructed fake data; the experiments are of the nature of local simulation and validation, and the results are used to examine the methodology and demonstrate the experimental process, and do not represent measurements of real production networks.

Table 2. Canary inventory

No.	Canary Content	Layer
1	Access key (AK/SK) in cloud instance metadata	Infrastructure layer
2	Database connection string echoed in training task logs	Infrastructure layer
3	Backup file in database backup bucket	Infrastructure layer
4	Knowledge base original documents	Infrastructure layer
5	Container service account (ServiceAccount) token	Infrastructure layer
6	Host machine shadow file hashes	Model application layer
7	Sensitive records in the user table	Platform component layer
8	Model weight file watermark	Platform component layer
9	Cross-tenant secret slices in the vector database	Platform component layer
10	Hardcoded key at debug endpoint	Model application layer
11	Client hardcoded key	Model application layer
12	Internal instructions in the system prompt	Model application layer
13	The user's own session in the client cache	Model application layer
14	Others' sessions exported through unauthorized access	Model application layer

Two engines are implemented on the attack side: the Kill Chain engine executes in sequence across the seven stages of reconnaissance, weaponization, delivery, exploitation, installation, command and control, and actions on objectives, with each stage following the exploitation methods corresponding to ATT&CK tactics, and the canaries touched being accumulated step by step to the final exfiltration phase.

The scan engine only executes harmless probing, confirming the existence of vulnerabilities and stopping there, without conducting deep exploitation or data acquisition.

On the protection side, three state switches—baseline, managed, and technical—and an alert engine are implemented, with attack requests and background requests recording alerts, false positives, and detection times throughout.

C. Experimental Results and Analysis

Experiment 1: comparison of Kill Chain-driven and scan-driven approaches. The execution results of the two methods in the same range under the same baseline protection state are shown in Table 3.

Table 3. Comparison of the two testing methods (baseline protection state)

Metric	Scan-driven	Kill Chain-driven
Confirmed vulnerability points	12 types (existence confirmed only)	12 (all deeply exploited)
Canaries touched	1/14 (7.1%)	14/14 (100%)
Attack phases completed	4/7	7/7
Data acquisition chain	None (not chained)	One complete chain
Alerts during the attack	0	2

The seven-stage execution record of the Kill Chain engine is as follows: the reconnaissance stage discovered API documentation and management plane exposure for seven services.

The weaponization stage preconstructed six categories of payloads including injection, unauthorized access, and traversal.

The delivery stage selected three entrances requiring no credentials.

The exploitation stage confirmed 12 vulnerability points, obtained cloud access keys through privilege escalation, and touched 12 canaries.

The installation stage completed persistence.

The command and control stage established an external communication channel.

The actions on objectives stage extracted and exfiltrated data in bulk, touching 14 canaries cumulatively.

From the execution record, the scan-driven approach

stops in the first half of the exploitation stage: it lists 12 categories of vulnerabilities, but because it stops upon confirming existence, it neither escalates privileges nor chains subsequent stages, ultimately touching only one canary (the object storage sample), and cannot answer the question of "how much data would be taken away".

The Kill Chain-driven approach chains the seven stages into a single chain. Entering through the management plane exposure entrance discovered by reconnaissance, it obtains cloud access keys through injection and unauthorized-access privilege escalation, acquires container service account tokens through lateral movement, and reads database connection strings through training task injection, thereby retrieving vector slices across tenants.

It finally completes bulk exfiltration through client-side horizontal privilege escalation export and session cache, reconstructing the complete data acquisition chain running from the server side through to the client side (Fig. 1).

The touch rate rises from 7.1% to 100%, verifying Argument 1: the phase and tactic division of Kill Chain and ATT&CK can directly serve as the organizational framework for the penetration testing of AI systems, and the evaluation focus should fall on the depth of data touch rather than the number of vulnerabilities.

Experiment 2: comparison of the three protection states. The execution results of the same attacker under the three protection states are shown in Table 4.

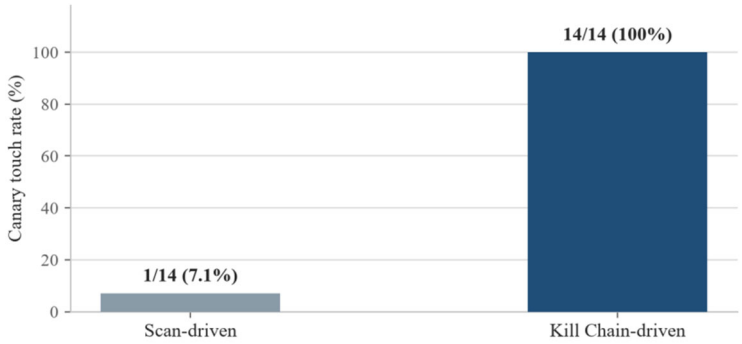

Fig. 1. Comparison of canary touch rates between the two testing methods.

Table 4. Comparison of the three protection states

Metric	Baseline	Managed	Technical
Canaries touched	14/14 (100%)	7/14 (50%)	1/14 (7.1%)
Layered touches (infrastructure / platform components / model application)	5/3/6	1/2/4	0/0/1
Attack surface reduction rate	-	60%	100%
Total alerts (false positive rate)	2 (0%)	14 (36%)	31 (13%)
Mean detection time (s)	Only canary exfiltration signal	0.049	0.089
Mean response time (s)	Not handled	No automatic handling	No high-risk triggered

Note: The mean detection time and mean response time are single-round measurements; millisecond-level differences are affected by request ordering and serve only as order-of-magnitude references.

The results can be interpreted from four aspects.

First, the managed state first eliminates "bulk leakage of large volume and broad scope": after privatizing bucket policies, adding tokens to metadata interfaces, isolating admin consoles, and disabling debug endpoints, infrastructure-layer touches fall from 5 to 1, the full-scenario touch rate falls from 100% to 50%, and the attack surface is reduced by 60%.

However, the managed state does not cover internal interface authentication, and the residual infrastructure-layer path is the unauthorized container orchestration interface, indicating that governance requires technical means for reinforcement.

Second, the technical state fills the gap of "precise exfiltration": unified gateway authentication clears the residual infrastructure-layer touches, while prompt filtering, retrieval-layer tenant filtering, and client data loss prevention further compress model-application-layer touches from 4 to 1, and the full-scenario touch rate from 50% to 7.1% (Figs. 2 and 3).

The two exhibit a ladder relationship of "management laying the foundation, technology closing the gap"—without the managed state, technical measures would have to be repeatedly constructed across countless entrances, and without the technical state, management requirements cannot be enforced as mandatory checkpoints, which confirms the division-of-labor logic of "macro emphasis on management, micro emphasis on technology" in Section III.

Third, the technical state still leaves one residual canary: the file read within the tool invocation whitelist lacks

parameter validation and can still read host machine files, indicating that there is no permanent technical defense line and that continuous observation, judgment, decision, and action according to the OODA loop are required.

Fourth, the managed state "can see but cannot handle": after log convergence, the attack period produces nine alerts with a mean detection time of 0.049 s, but there is no automatic response, and after high-risk exfiltration occurs isolation is not achieved, with no value for mean response time, which argues that the response and recovery phases of IPDRR must be technically implemented, and that detection capability can form a closed loop only when accompanied by handling capability.

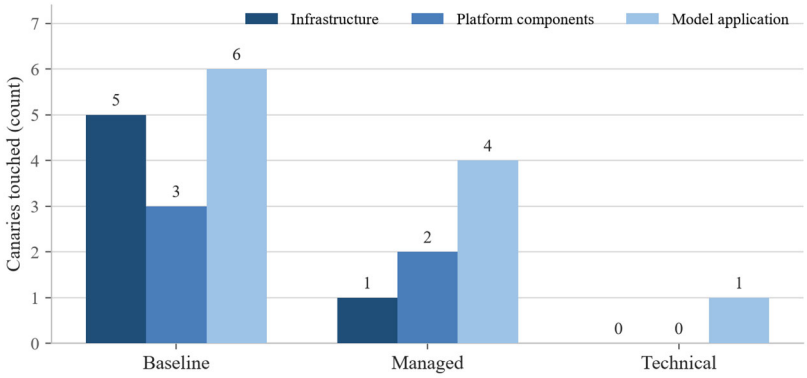

Fig. 2. Number of layered canary touches under the three protection states.

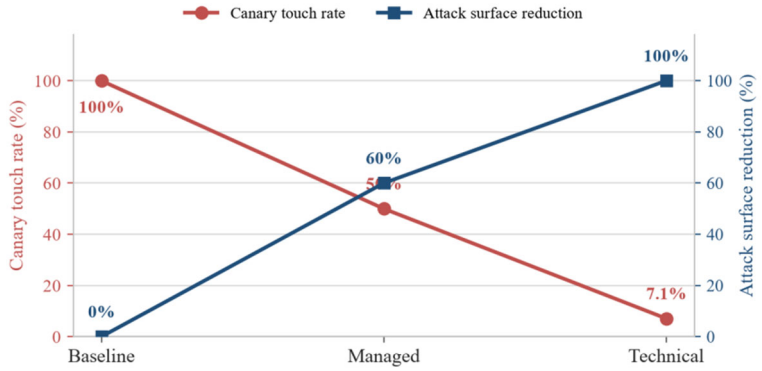

Fig. 3. Canary touch rate and attack surface reduction rate under the three protection states.

D. *Experimental Conclusions and Limitations*

The two sets of experiments verify the two arguments of this paper respectively. Argument 1 is verified: Kill Chain and ATT&CK can directly organize the penetration testing of AI systems, reconstructing the complete data acquisition chain along the three layers, with a canary touch rate of 100%, far higher than the scan-based 7.1%.

Argument 2 is verified: management measures first eliminate bulk leakage, technical measures then compress precise exfiltration, the two exhibiting a ladder relationship rather than a substitution relationship, and detection and automatic response must be complete to close the loop.

At the methodological level, "depth of data touch as the criterion" is operationalized in these experiments as the canary touch rate, and it is recommended for continued use as a standardized metric.

Four limitations should be noted: first, the experiments involve a single range, a single attacker, and a single round, without multiple repetitions or double-blind design.

Canary coverage cannot be complete, and the touch rate is a lower-bound estimate.

Second, the range is a simulated system, and the difficulty of vulnerability exploitation is lower than in real environments.

Third, the three protection states are switched instantaneously, without simulating the human delays in the implementation of management measures.

Fourth, the false positive rate is based on small-sample background requests, and its absolute value is for reference only..

V. Conclusion

In recent years, artificial intelligence has developed explosively, with exploratory applications in various industry sectors, and in some sectors it has already played an important role. AI security is a security risk that has been frequently mentioned and highly valued since the emergence of AI, posing an expansion requirement for established security methodologies.

In summary, the changes brought by AI systems lie in the expansion of asset forms, data distribution, and attack surfaces, while the underlying logic of traditional penetration testing models and security protection models still holds.

This paper further conducts two sets of controlled experiments in a locally simulated range: the canary touch rate of Kill Chain-driven testing reaches 100%, while the ladder protection of "management laying the foundation, technology closing the gap" compresses the touch rate to 7.1%, supporting the above judgment at the experimental

level.

Conflict of Interest

The authors declare no conflict of interest.

References

[1] The Cyber Express. (2025). Cybersecurity 2026: Why Protecting Data Matters More Than Stopping Attacks[EB/OL]. [Online]. Available: https://thecyberexpress.com/cybersecurity-2026-and-data-centric-security/

[2] Verizon. (2026). Data Breach Investigations Report (19th Edition). [Online]. Available: https://www.verizon.com/business/resources/reports/dbir/

[3] E. M. Hutchins, M. J. Cloppert, R. M. Amin. (2011). Intelligence-Driven Computer Network Defense Informed by Analysis of Adversary Campaigns and Intrusion Kill Chains. *Leading Issues in Information Warfare & Security Research*. [Online]. Available: https://www.lockheedmartin.com/content/dam/lockheed-martin/rms/documents/cyber/LM-White-Paper-Intel-Driven-Defense.pdf

[4] MITRE. ATT&CK Framework[EB/OL]. [Online]. Available: https://attack.mitre.org

[5] NIST. (2018). Framework for Improving Critical Infrastructure Cybersecurity, Version 1.1[R]. [Online]. Available: https://nvlpubs.nist.gov/nistpubs/CSWP/NIST.CSWP.04162018.pdf

[6] NIST. (2023). Artificial Intelligence Risk Management Framework (AI RMF 1.0): NIST AI 100-1[R]. [Online]. Available: https://nvlpubs.nist.gov/nistpubs/ai/NIST.AI.100-1.pdf

[7] ITU-T. (2016). X.1642: Guidelines for the Operational Security of Cloud Computing[Z]. [Online]. Available: https://www.itu.int/rec/T-REC-X.1642-201603-I

[8] MITRE. ATLAS: Adversarial Threat Landscape for Artificial-Intelligence Systems[EB/OL]. [Online]. Available: https://atlas.mitre.org

[9] OWASP. Top 10 for Large Language Model Applications[EB/OL]. [Online]. Available: https://owasp.org/www-project-top-10-for-large-language-model-applications

[10] National Information Security Standardization Technical Committee. (2019). GB/T 22239—2019 Information Security Technology: Baseline for Classified Protection of Cybersecurity[S]. *Beijing: Standards Press of China.* [Online]. Available: https://openstd.samr.gov.cn/bzgk/std/newGbInfo?hcno=BAFB47E8874764186BDB7865E8344DAF

[11] National People's Congress. (2021). Data Security Law of the People's Republic of China[Z]. [Online]. Available: http://www.npc.gov.cn/englishnpc/c2759/c23934/202112/t20211209_385109.html

[12] National People's Congress. (2021). Personal Information Protection Law of the People's Republic of China[Z]. [Online]. Available: https://en.spp.gov.cn/2021-12/29/c_948419.htm

[13] Cyberspace Administration of China *et al.* (2023). Interim Measures for the Administration of Generative AI Services[Z]. [Online]. Available: https://www.cac.gov.cn/2023-07/13/c_1690898327029107.htm

Copyright © 2024 by the authors. This is an open access article distributed under the Creative Commons Attribution License which permits unrestricted use, distribution, and reproduction in any medium, provided the original work is properly cited (CC BY 4.0).