

THREAT RESEARCH · PROMPT INJECTION · PERSISTENT AI MEMORY

The Inception Hack

How a honeypot article can install a dormant, nested prompt injection inside an AI “second brain.”

A concealed first-stage prompt teaches the model how to decode a second instruction distributed through the visible prose. The victim supplies persistence by voluntarily saving the article into their own knowledge vault.

“The attacker does not hide the final instruction. They hide the instruction that teaches the AI how to create it.”

Responsible research notice

READ FIRST

This paper describes an attack *architecture* at a defensive level of abstraction. It intentionally excludes deployable malicious payloads, operational exfiltration instructions and weaponised carrier text. The only worked example decodes to the inert phrase “this is only a test.” All testing described here assumes benign payloads, synthetic data, sandboxed tools and coordinated disclosure to affected vendors. The Inception Hack is presented as a *proposed* attack class and research programme: its components are grounded in established prompt-injection and retrieval risks, but the reliability of the full nested lifecycle must be established empirically.

CONTENTS

What this paper covers

Executive summary · 01	10 · Variants & evolution paths · 08
1 · Scope, status & terminology · 02	11 · Why common defences may miss it · 09
2 · The rise of the AI second brain · 03	12 · Threat model · 10
3 · Why ordinary prompt injection isn't the whole story · 03	13 · Strategic defence framework · 10
4 · Anatomy of the nested attack · 04	14 · Proposed experimental evaluation · 12
5 · The honeypot inversion · 05	15 · Limitations & falsifiability · 12
6 · Concealment vs construction: $P = D(K,C)$ · 06	16 · Relationship to existing attack classes · 13
7 · Authority laundering · 06	17 · Implications for builders, vendors & users · 13
8 · Lifecycle, dormancy & vault infection · 07	18 · Conclusion & the Sentinel position · 14
9 · Impact tiers · 08	References · Appendix A: exposure checklist · 15

The most dangerous prompt in a second brain may be the one that was never written

A new generation of personal and enterprise AI systems — commonly described as *AI second brains* — connects large language models to persistent stores of notes, saved articles, emails, project files and private documents. This architecture promises continuity and context. It also creates a security condition that conventional document scanning was never designed to manage.

The Inception Hack is a proposed nested, delayed-action prompt-injection technique designed for these persistent knowledge systems. The delivery vehicle is a genuinely useful *honeypot article* that a victim voluntarily saves into an AI-accessible vault. Concealed within it is a short first-stage prompt — potentially white-on-white text or another visually suppressed layer. That first-stage prompt does not contain the malicious objective. It tells the model how to extract and assemble a second instruction distributed through the article's *visible* prose.

The visible text is therefore not merely cover. It is a *semantic carrier*. Each selected word may be ordinary, correctly spelled and contextually appropriate. The final instruction appears only when the model applies the concealed decoding rule. The AI becomes the first location in which the complete payload exists.

CORE FINDING

The attack separates **activation**, **payload** and **persistence**. The hidden pre-prompt supplies the decoder; the visible article supplies the fragments; the victim's vault supplies persistence; and the AI supplies runtime assembly and interpretation. Two security concepts follow: the **instruction-existence gap** (static inspection happens before the complete operational instruction exists) and **authority laundering** (untrusted, model-derived text is promoted into an instruction-capable control plane because a trusted model produced it).

This paper does not claim that white text, acrostics, steganography or indirect prompt injection are individually new. The proposed contribution is the *integrated threat pattern*: a nested decoder-and-carrier construction, deliberately packaged as high-value content for voluntary ingestion into persistent AI memory, with delayed and configuration-dependent activation.

KEY CONCLUSIONS

- AI second brains create *durable* exposure because untrusted content can be stored, repeatedly retrieved and processed alongside private data.
- The first-stage hidden prompt is essential: it gives the model a reason and a method to reconstruct the otherwise inert carrier.
- The final payload need not exist as a contiguous source string; it may emerge only after model-led transformation.
- The same document can be low-impact in a read-only vault and materially dangerous in an agentic system with email, browser, file or API access.
- Defence must combine presentation-layer sanitisation with provenance, instruction/data separation, least privilege, action authorisation and runtime monitoring.

Scope, status & terminology

The Inception Hack is presented as a proposed attack class and research programme. Its components are grounded in established prompt-injection and retrieval risks, but the reliability of the complete nested lifecycle must be measured empirically. The paper therefore separates architectural reasoning from demonstrated capability.

WORKING DEFINITION

The Inception Hack — a vault-borne, nested indirect prompt injection in which a concealed first-stage decoder prompt causes an AI system to reconstruct a second-stage instruction from visible, apparently benign carrier content that the victim has voluntarily saved into persistent memory.

TERM	MEANING
Honeypot article	A useful, credible document engineered to be saved, indexed or shared by the target audience.
Hidden pre-prompt	The concealed first-stage instruction specifying how the carrier should be decoded or interpreted.
Semantic carrier	The visible prose containing distributed words or fragments from which the second-stage instruction is reconstructed.
Runtime payload	The complete instruction produced after the model applies the decoder to the carrier.
Instruction-existence gap	The interval in which the source components exist but the complete operational instruction has not yet been assembled.
Authority laundering	The elevation of untrusted content into apparent instruction authority after transformation by a trusted model.
Vault infection	Persistent or propagating influence inside a knowledge system, including repeated activation or contamination of derived notes and memory.
Configuration-dependent effect	The principle that impact is determined by the target's retrieval, memory, permissions, tools and confirmation model.

WHAT IS & IS NOT CLAIMED

This paper does *not* claim that positional encoding, white-text concealment, indirect prompt injection or persistent-memory poisoning are individually unprecedented, nor that every model will decode, adopt or execute a reconstructed instruction. The claimed novelty is the *threat architecture and target selection*: concealed decoder logic nested with a visible carrier, delivered through content worth saving, activated inside an AI-accessible vault, and amplified by persistence and connected capabilities.

SECTION 2

The rise of the AI second brain

AI second brains are moving from niche experiments into mainstream use. A typical system combines a vault of personal documents, a retrieval-augmented generation (RAG) pipeline, embeddings or semantic search, a local or hosted language model, automated summaries, persistent memory, and connections to email, calendars, browsers or task systems — increasingly driven by autonomous or semi-autonomous agents.

The appeal is obvious. Instead of asking an AI isolated questions, the user grants it the accumulated context of their life or organisation. The system becomes research assistant, personal archive, analyst, planner and

operator. But this introduces a profound trust problem: the same assistant is encouraged to consume large amounts of *external* material while being trusted with *private* information and *operational* capability.

In conventional computing, data and executable instructions occupy recognisably different layers, processed under explicit execution rules. In many AI second brains, developer policy, user intent, retrieved evidence, quoted text, tool output and generated reasoning all arrive as natural language. The system must *infer* which language describes the task and which language attempts to redefine it. That ambiguity is the opening.

SECTION 3

Why ordinary prompt injection isn't the whole story

Most discussions of indirect prompt injection focus on a straightforward pattern: an attacker places an instruction inside content an AI later reads — white text on a white background, tiny or off-screen text, commands in metadata, text concealed in images. The principle is simple: a human sees one thing while the AI sees an additional instruction.

These attacks are real, but they share a *structural weakness*: the malicious instruction exists somewhere in recognisable form. It may be hidden, disguised or encoded, but at some point the complete command is physically present. That gives defenders something concrete to search for — strip invisible formatting, normalise text, extract hidden layers, flag imperative language, classify the document as instruction-bearing.

The Inception Hack changes the structure. The hidden content does not contain the final malicious instruction. It contains only the instructions required to discover it.

Anatomy of the nested attack



Figure 1. The complete attack separates social delivery, hidden activation, visible carrier content, persistent storage, runtime construction and downstream authority. No single stage contains the whole malicious operation.

Stage one — the hidden pre-prompt

A short instruction is concealed from ordinary human viewing while remaining available to the ingestion pipeline or model. White-on-white text is the clearest illustration because it demonstrates the asymmetry directly: the human sees an article; the machine-readable layer contains an additional instruction.

Equivalent channels include suppressed HTML, off-page or zero-size text, comments, metadata, alt-text, OCR-confusable layers or document-conversion artefacts. Crucially, the pre-prompt is a *decoder*, not the payload: it may tell the model to select particular positions, follow a sequence, map headings to words or combine fragments. Without it, the carrier is merely an improbable arrangement of prose. The hidden prompt gives the model a reason and a method to read the document adversarially.

Stage two — the visible semantic carrier

The visible article contains the components of the second instruction, distributed across hundreds or thousands of words. Each selected term can be ordinary and locally appropriate; the surface article remains coherent, informative and genuinely worth preserving. This is what distinguishes a carrier from conventional hidden text: the payload is not rendered invisible — its components are visible, but their *operational relationship* is concealed. The key carries the method; the article carries the material; the model performs the composition.

A deliberately harmless illustration

To show the structure without publishing an operational attack, consider a benign carrier in which every fifth word has been deliberately chosen (positions 5, 10, 15, 20, 25 are highlighted):

My favourite part of **this** slow morning routine really **is** simple and costs nothing: **only** fresh coffee ground by **a** hand mill before the **test**.

DECODES TO → **"THIS IS ONLY A TEST"**

Read normally, it is a throwaway sentence about coffee. The hidden pre-prompt contains no harmful payload; the decoded result appears only after the model performs the extraction. In a real Inception Hack the carrier would be far longer and more natural — a detailed, high-quality article spanning several thousand words, where the cadence is impossible to notice by eye. The better the writing, the stronger the delivery mechanism. (This inert, self-describing construction is the prompt-injection analogue of the EICAR antivirus test file: it proves the channel exists while being incapable of harm.)

Stages three to five — ingestion, assembly, effect

ERIC / SENTINEL / AI SECURITY RESEARCH / THE INCEPTION HACK / PUBLIC

The content is designed for the target community; a sophisticated article on AI architecture, investing, law or security is more effective than obvious spam because it is likely to be downloaded, cited, clipped into notes or indexed by an autonomous research process. Activation may occur during initial ingestion, later summarisation, semantic retrieval, re-indexing, scheduled research or cross-document synthesis. When the pre-prompt and carrier enter the same effective context, the model applies the decoding instruction and generates the second-stage payload. The *effect* is then configuration-dependent — the payload is only one component of risk; capability architecture determines blast radius.

SECTION 5

The honeypot inversion

The article is not merely a container; it is part of the attack strategy. The attacker creates content the target genuinely wants to save: original, well-researched, highly relevant to an emerging field, framed as a valuable technical reference, optimised to appear in research and recommendation workflows.

This inverts conventional malware delivery. The victim is not tricked into opening a suspicious attachment — they deliberately place the document inside their most trusted knowledge system because the content is valuable.

The article crosses the boundary through relevance rather than deception alone. It may be bookmarked, downloaded, copied into a notes vault, indexed by an assistant, included in a research corpus, summarised, cited, or imported into a company knowledge base. The attacker does not have to breach the vault. Relevance becomes the delivery mechanism; persistence is supplied by the product itself.

Concealment versus construction

This is the defining feature. Traditional hidden prompt injection conceals an instruction that *already exists*. The Inception Hack causes the model to *construct* an instruction that does not yet exist in complete form.

$$P = D(K, C)$$

P is not contained in K · P is not contained in C · yet $D(K, C) = P$

K = hidden pre-prompt (decoding key) · C = visible carrier · D = model-led decoding · P = reconstructed payload

The notation is conceptual, not cryptographic. Language models do not decode with deterministic precision — they infer patterns, resolve ambiguity, repair omissions and interpret semantic equivalents. This flexibility makes the channel more *tolerant*, but also makes success dependent on model family, tokenisation, formatting, context, sampling and safety policy. The model becomes the first place where the malicious instruction exists as a complete semantic object. *That is the inception: the idea is not simply hidden from the reader — it is formed inside the machine.*

THE INSTRUCTION-EXISTENCE GAP

Before decoding, static controls see a suspicious-but-generic extraction rule and an apparently harmless article. After decoding, the system has *already performed* the transformation requested by untrusted content. The complete instruction appears at runtime, inside the very component expected to evaluate it. A scanner cannot be the only control when the object it is meant to detect may not exist until after static inspection.

SECTION 7

Authority laundering: the deeper vulnerability

The decoder is not the ultimate security failure. The failure occurs when output derived from an untrusted source is allowed to influence the control plane as though it came from the user, developer or system policy. The model appears trusted because it is the application's reasoning engine — and that trust inadvertently transfers to text the model has extracted or reconstructed. The system launders the source's authority through its own generation process.



Figure 2. The decisive failure: transformation increases authority. Untrusted source material becomes derived text and is then treated as a control command.

SECURITY INVARIANT · THE INHERITANCE RULE

Transformation must not increase authority. Text extracted, decoded, summarised, translated or inferred from untrusted content must retain the trust classification of its source. A reconstructed instruction is **quoted data, not a control command** — regardless of whether the model generated it correctly. This principle applies far beyond the Inception Hack: to indirect prompt injection generally, tool output, web content, emails, attachments, OCR, transcriptions and model-generated summaries of those sources.

Attack lifecycle, dormancy & vault infection

PHASE	DESCRIPTION
1 · Targeting	Identify communities using personal or enterprise knowledge vaults and the topics they consider worth preserving.
2 · Honeypot production	Create a credible, useful, discoverable article whose surface content survives genuine scrutiny.
3 · Pre-prompt concealment	Insert a short decoder instruction into a machine-readable but visually suppressed layer.
4 · Carrier engineering	Arrange selected visible words so the decoder can reconstruct a second-stage instruction.
5 · Distribution	Publish through ordinary channels: search, newsletters, repositories, professional networks, community sharing.
6 · Voluntary ingestion	The victim downloads, clips, saves, indexes or imports the article into a vault.
7 · Dormancy	The article persists without immediate effect — through backups, synchronisation and derived indexes.
8 · Retrieval / reprocessing	A query, scheduled process, re-index or topic synthesis returns the document to active context.
9 · Runtime construction	The hidden pre-prompt activates decoding; the model assembles the runtime payload.
10 · Authority confusion	The system fails to preserve source trust and treats the derived text as operationally relevant.
11 · Effect	The payload biases reasoning, contaminates memory or attempts a tool-mediated action.
12 · Propagation	Generated notes, summaries or repeated retrieval preserve or amplify the influence.

Dormancy describes a stored carrier that activates only when retrieved. **Vault infection** describes a stronger state in which attacker influence persists beyond the original file — through generated summaries, linked notes, memory entries, retrieval priorities or automated reports. The term should not be confused with a self-replicating computer virus; it denotes persistent knowledge contamination and recurrent behavioural influence. Where hostile influence propagates into derived memory and affects multiple future workflows, *knowledge-vault infection* is a reasonable description of the security state.

SECTION 9

Impact tiers

The same carrier produces radically different outcomes in different systems. Impact should be assessed against capability, not payload alone.

Tier 0	Inert	The document is stored but never enters a model context.
Tier 1	Interpretive	The decoded text changes a summary, answer, ranking or recommendation.
Tier 2	Persistent	The influence is written into memory, summaries, linked notes or future context.
Tier 3	Privileged	The system accesses private information outside the user's immediate request.
Tier 4	Operational	The system attempts external actions through email, files, browsers or APIs.
Tier 5	Propagating	Derived artefacts carry the influence into additional systems or users.

SECTION 10

Variants & evolution paths

VARIANT	DESCRIPTION
Positional carrier	Every nth word, sentence, heading-adjacent token or table coordinate — any deterministic position.
Semantic carrier	Words selected by category, meaning or relationship rather than exact position.
Cross-document assembly	Key, carrier and activation condition stored in separate documents and reunited by retrieval.
Multistage decoding	The first reconstruction produces another rule rather than the final payload.
Multimodal composition	Fragments distributed across text, images, diagrams, audio, OCR or structured data.
Conditional activation	The derived instruction is relevant only when a user, project, date, tool or private datum is present.
Capability-latent carrier	Stored while harmless, then actionable after the user later connects new tools.
Derivative propagation	Summaries or generated notes preserve enough of the key/carrier to reactivate later.

RESEARCH CAUTION

These are plausible variants, not demonstrated claims. A credible publication must distinguish threat modelling from measured attack reliability. The *cross-document* variant is especially notable: individually each document may appear harmless; the attack becomes visible only when the retrieval system composes them into one context — the second brain’s core capability becomes part of the exploit.

Why common defences may be insufficient

CONTROL	RESIDUAL EXPOSURE
Hidden-text stripping	Highly relevant, and capable of defeating the simplest pre-prompt — but only when <i>all</i> presentation layers are reliably extracted and sanitised before model access.
Keyword / imperative scanning	The carrier contains ordinary vocabulary; the pre-prompt can use generic extraction language rather than the harmful objective. No complete malicious command exists to match.
Per-document classification	The dangerous property may exist only in the relationship between hidden and visible layers — or across several retrieved objects.
Human editorial review	Readers evaluate meaning and quality; they rarely inspect rendering layers or count periodic words unless already prompted to suspect it.
Model-based injection classifiers	A classifier may score well on explicit instructions yet receive an input view that does not contain the final reconstructed payload.
Prompt delimiters	Useful but not a hard security boundary. The same model still interprets instructions and untrusted evidence inside one reasoning process.
One-time ingestion checks	Persistent systems repeatedly retrieve content, and capability changes can alter the risk of already-stored material.

SECTION 12

Threat model

ATTACKER ASSUMPTIONS

- Can publish or influence content likely to be saved or retrieved by the target community.
- Controls the article's visible prose and at least one machine-readable presentation layer.
- May infer likely models, formats, retrieval topics or connected tools, but need not access the victim's vault directly.
- Does not require knowledge of the victim's private data; the system may bring that data into context after ingestion.

NECESSARY TARGET CONDITIONS

- Untrusted content reaches a model without complete visual-layer sanitisation.
- The model can perform the decoder transformation over the carrier.
- Derived content can influence subsequent reasoning, memory or tool planning.
- The application lacks a reliable provenance and authority boundary.
- Consequential actions are possible without independent validation or confirmation.

SECTION 13

Strategic defence framework

Because the payload has no fixed location, defence cannot be “find the bad string.” It must be structural.

1. **Canonicalise the human-visible document.** Before ingestion, reconcile what the human sees with what the machine can extract. Detect white-on-white text, opacity, zero-size fonts, off-page

objects, hidden HTML, comments, metadata, alt-text, OCR discrepancies and unusual PDF layers.

Where extracted text materially differs from the rendered page, quarantine or surface the discrepancy.

2. **Detect decoder-like behaviour.** Flag content that instructs the model to select, count, reorder, decode or reinterpret other content — especially when the instruction originates inside retrieved material. Not proof of attack, but a risk signal that should alter processing and permissions.
3. **Enforce instruction provenance.** Every instruction-capable element should carry a traceable origin and authority level: developer policy, authenticated user request, untrusted retrieval, tool output or model-derived interpretation. Never collapse these into undifferentiated text.
4. **Preserve trust through transformation.** Extracted or generated content inherits its source's trust level. A decoded phrase from an article remains untrusted data.
5. **Separate extraction, reasoning and execution.** Extraction should produce schema-bound data, not free-form instructions. A separate planner evaluates the authenticated user objective; a policy layer validates any proposed action; a tool executor receives only authorised parameters.
6. **Bind actions to explicit user intent.** Before consequential execution, confirm the action is causally supported by the user's request rather than by retrieved content. Block when purpose, destination or data scope cannot be derived from authenticated intent.
7. **Apply least privilege & capability isolation.** A summariser should not inherit email, deletion or publishing capability. Grant tools narrowly, temporarily and per task; separate read from write, internal retrieval from outbound communication.
8. **Require meaningful confirmation.** For sending, sharing, purchasing, deleting, publishing, permission changes or sensitive disclosure, show the action, parameters, destination, data involved and source lineage. Generic "continue?" prompts are insufficient.
9. **Monitor persistent-memory writes.** Treat memory updates as consequential. Record which sources influenced the write, whether hidden text was present and whether it contains instruction-like content. Provide rollback and quarantine.
10. **Reassess the vault when capabilities change.** Connecting a new tool changes the threat model of every existing document. Rescan or constrain old content rather than silently granting it influence over new capabilities.

Proposed experimental evaluation

The concept should be converted into a reproducible, *non-weaponised* benchmark. The objective is not a deployable exploit but a measurement of where the architecture fails.

METRIC	QUESTION
Decoder recognition rate	Does the model detect and follow the hidden first-stage extraction rule?
Reconstruction accuracy	Does it correctly assemble the inert test phrase from the carrier?
Instruction-adoption rate	Does it treat the reconstructed phrase as an instruction rather than quoted data?
Persistence rate	Is the influence written into memory or derived artefacts?
Action-attempt rate	In a sandbox, does the system propose or invoke the simulated tool action?
Detection rate	Which ingestion, classifier, policy or runtime control identifies the sequence?
User-visibility rate	Does the interface disclose that hidden text or unexpected instructions influenced the output?

Independent variables: model and version; document format (HTML, DOCX, PDF, Markdown, image-derived OCR); visibility mechanism and sanitisation pipeline; carrier length, density and naturalness; exact-positional vs semantic decoding; retrieval chunking and key/carrier separation; initial ingestion vs delayed retrieval; persistent memory on/off; tool permissions and confirmation policy; single- vs cross-document composition.

SAFE TEST DESIGN

Use benign payloads such as “record that this is a controlled test” and synthetic records with no real secrets. Mock all external tools. Success criteria must distinguish *decoding* from *adoption* and adoption from *execution*. Decoding a phrase is not proof of compromise — the critical test is whether the system grants the reconstructed phrase operational authority. Provide coordinated disclosure where a test reveals product-specific weakness.

SECTION 15

Limitations & falsifiability

A high-quality threat paper must be explicit about what could disprove or narrow its thesis.

- Models may refuse hidden instructions, treat decoded output as quoted content, or fail positional extraction reliably.
- Robust document canonicalisation may remove the pre-prompt before ingestion.
- Retrieval chunking may separate the key from enough carrier content to prevent reconstruction.
- Natural carrier construction may become statistically detectable or degrade article quality.
- Action policies may prevent any meaningful effect even when decoding and adoption occur.
- Cross-model transfer may be poor, requiring model-specific carrier design.
- The attack may prove primarily a red-team edge case rather than a scalable real-world technique.

These possibilities do not weaken the need for testing; they *define* it. The paper’s strongest contribution is a precise, falsifiable claim about a neglected interaction between hidden activation, distributed payloads and persistent AI memory.

Relationship to existing attack classes

ADJACENT CLASS	DISTINCTION
Direct prompt injection	The attacker controls the user-facing prompt. The Inception Hack arrives through content the user asks the system to read.
Indirect prompt injection	The parent class: adversarial instructions in external data. This technique adds nested decoding and persistent-vault delivery.
Hidden-text injection	Conceals a complete instruction in a suppressed layer. Here the hidden layer is principally the <i>decoder</i> , not the objective.
Steganography / acrostics	Provide carrier mechanisms. The novelty lies in model-mediated instruction construction and authority confusion inside an agentic system.
RAG poisoning	Manipulates material likely to be retrieved. The honeypot model adds voluntary ingestion and long-term persistence in user-controlled vaults.
Memory poisoning	Persists adversarial influence in memory. The Inception Hack describes one delivery and activation chain that may lead to that state.
Prompt worms	Attempt propagation between systems. Vault infection may create preconditions for propagation, but self-replication is not assumed.

SECTION 17

Implications for builders, vendors & users

For homebrew builders

Do not place raw retrieved content into an instruction-capable prompt and assume delimiters make it inert. Build an ingestion boundary, canonicalise files, tag provenance, isolate transformations and require confirmation for writes and external actions.

For enterprise vendors

Treat knowledge ingestion as a supply-chain boundary. Security claims should cover hidden presentation layers, derived instructions, cross-document composition, memory writes, tool lineage and capability changes — not merely explicit prompt strings.

For security teams

Extend prompt-injection testing beyond jailbreak phrases: nested transformations, delayed retrieval, persistent memory, tool-mediated effects, and content genuinely useful enough to be retained.

For users

A trusted publication source does not guarantee that every machine-readable layer is safe. Review what your system ingests, restrict tool permissions, avoid autonomous execution from retrieved content, and understand whether your vault can write to persistent memory without approval. The most dangerous configuration combines untrusted content, private data, persistent memory and an external action channel.

Conclusion

We have been defending against hidden *instructions*. The Inception Hack suggests we must also defend against hidden instruction *assembly*. A honeypot article enters the system because it is useful. A concealed pre-prompt tells the model how to read it. The visible prose supplies the pieces of a second prompt. The model assembles the payload inside its own context. The vault provides persistence; the user's data and connected tools provide consequence. No breach is required, and no obviously malicious attachment — the victim installs the document voluntarily, and the AI performs the final construction itself.

The final instruction may not exist as a complete sentence in any source layer. That creates an *instruction-existence gap* for static controls and a risk of *authority laundering* when the system mistakes its own derived text for legitimate control. The future of prompt injection may therefore involve fewer explicit commands: attackers may supply fragments, keys, rules and distributed cues, and let the model supply the missing interpretation.

THE GOVERNING PRINCIPLE

What an AI system reads must never automatically become what it obeys — and what it derives from untrusted content must never acquire more authority than the source possessed. Until that boundary is enforced, the most dangerous prompt inside an AI second brain may be the one that was never actually written.

THE SENTINEL POSITION

AI security begins where information becomes instruction

EAIC Sentinel evaluates whether untrusted information can become operational control inside AI knowledge systems, retrieval pipelines and agentic workflows. An Inception exposure assessment examines the complete lifecycle rather than searching only for known malicious phrases:

- Human-visible vs machine-readable document comparison
- Hidden-layer extraction & canonicalisation
- Decoder-language & structural-carrier detection
- Retrieval & chunking trust boundaries
- Instruction provenance & authority inheritance
- Cross-document composition exposure
- Persistent-memory write controls & rollback
- Tool permission scope & action confirmation
- Outbound communication & data-loss channels
- Forensic lineage from source document to tool call

Book an Inception exposure assessment for your AI stack → [LANDING PAGE URL]

Selected references & research context

Greshake, K., Abdelnabi, S., Mishra, S., Endres, C., Holz, T. & Fritz, M. (2023). *Not What You’ve Signed Up For: Compromising Real-World LLM-Integrated Applications with Indirect Prompt Injection*. arXiv:2302.12173.

Zhan, Q., Liang, Z., Ying, Z. & Kang, D. (2024). *InjecAgent: Benchmarking Indirect Prompt Injections in Tool-Integrated Large Language Model Agents*. arXiv:2403.02691.

Willison, S. (2022–2025). “*Prompt injection*” series and “*The lethal trifecta for AI agents*” (private data + untrusted content + exfiltration channel). simonwillison.net.

OWASP Foundation. *OWASP Top 10 for Large Language Model Applications* — LLM01: Prompt Injection.

National Institute of Standards and Technology. *AI Risk Management Framework (NIST AI 100-1)* and *Generative AI Profile (NIST AI 600-1)*.

NOTE ON SOURCING

References establish the broader indirect-prompt-injection and agent-security context. The specific nested honeypot-vault architecture advanced here remains a *proposed* research contribution requiring empirical validation. Two future-dated citations present in an earlier draft were removed pending verification; add any peer-reviewed sources you can independently confirm.

Second-brain exposure checklist

AREA	ASSESSMENT QUESTION	STATUS / EVIDENCE
Ingestion	Does the pipeline compare rendered content with extracted machine-readable text?	
Ingestion	Are hidden text, metadata, comments and alternate layers stripped or quarantined?	
Retrieval	Is retrieved content explicitly marked and isolated as untrusted data?	
Retrieval	Can key and carrier fragments from separate documents enter one context?	
Transformation	Can a retrieved document instruct the model to decode or reorder other content?	
Authority	Do model-derived outputs inherit source trust classification?	
Memory	Can retrieved content create or modify persistent memory without review?	
Tools	Are permissions scoped per task and separated between read and write?	
Actions	Must consequential actions be justified by the authenticated user request?	
Confirmation	Does approval show destination, data scope, parameters and source lineage?	
Monitoring	Can investigators trace a tool call back to documents and transformations?	
Change	Are stored documents reassessed when new capabilities are connected?	