iJRASET
International Journal For Research in Applied Science and Engineering Technology

INTERNATIONAL JOURNAL FOR RESEARCH

IN APPLIED SCIENCE & ENGINEERING TECHNOLOGY

Volume: 13 **Issue:** XI **Month of publication:** November 2025

DOI: https://doi.org/10.22214/ijraset.2025.75690

www.ijraset.com

Call: 08813907089 | E-mail ID: ijraset@gmail.com

International Journal for Research in Applied Science & Engineering Technology (IJRASET)
ISSN: 2321-9653; IC Value: 45.98; SJ Impact Factor: 7.538
Volume 13 Issue XI Nov 2025- Available at www.ijraset.com

The Shadow-Agentic Supply Chain: Empirical Analysis of Serialization Vulnerabilities and Autonomous Threat Vectors

Aditya Chordia
Independent AI security researcher

Abstract: ***The integration of Artificial Intelligence (AI) into the enterprise has precipitated a fundamental transformation in the cybersecurity threat landscape, characterized by the emergence of the "Shadow-Agentic Supply Chain." This phenomenon represents the dangerous convergence of "Shadow AI, the unsanctioned deployment of probabilistic models within organizational networks, and "Agentic AI," which endows these systems with the autonomy to execute multi-step workflows, manipulate external tools, and interact with sensitive data repositories without human intervention. This research paper presents an exhaustive empirical analysis of the security implications arising from this convergence, specifically focusing on the systemic serialization vulnerabilities inherent in the Python ecosystem that underpins modern Machine Learning (ML) infrastructure. Through a rigorous multi-phased investigation, comprising a controlled "Sleepy Pickle" injection experiment, a live "Snapshot" analysis of the Hugging Face and GitHub ecosystems, and a theoretical critique of the "Defender's Gap," we demonstrate that the prevailing trust models in AI supply chains are fundamentally broken.***

We provide empirical evidence that the pickle serialization format, currently the de facto standard for ML model distribution, permits the embedding of arbitrary code execution payloads that successfully evade traditional Endpoint Detection and Response (EDR) and network perimeter defenses. Furthermore, our ecological assessment reveals a systemic lack of security hygiene, quantified by the proliferation of undocumented models and the rampant exposure of cryptographic secrets necessary for agentic operations. By synthesizing these empirical findings with a detailed analysis of "Morris II" worm mechanics and "PoisonGPT" supply chain insertion techniques, we propose a novel "Agentic Kill Chain." This paper argues that the transition from passive predictive models to active autonomous agents necessitates a paradigmatic shift in supply chain security, moving beyond static vulnerability scanning toward behavioural attestation, cryptographic provenance, and the runtime confinement of agentic workflows.

Keywords: AI Security, Shadow AI, Supply Chain Security, Adversarial Machine Learning, Serialization Vulnerabilities, Agentic AI.

I. INTRODUCTION

The rapid and often unchecked adoption of Generative AI (GenAI) and Large Language Models (LLMs) has fundamentally altered the operational fabric of the modern enterprise, introducing a new class of security risks that transcend traditional software vulnerabilities. Organizations are aggressively integrating these technologies to drive automation and innovation, yet this adoption curve has significantly outpaced the development of robust security governance frameworks. The result is the widespread proliferation of "Shadow AI", unmanaged, unmonitored, and often insecure AI deployments that operate entirely outside the purview of central Information Technology (IT) and security teams.1 Unlike traditional "Shadow IT," which typically involves the unauthorized use of SaaS applications or cloud storage for static data, Shadow AI introduces dynamic, probabilistic entities capable of decision-making. When these shadow deployments are empowered with agentic capabilities, allowing them to browse the web, execute code, and interact with internal Application Programming Interfaces (APIs), they evolve into "Shadow Agents." These autonomous entities possess the capability to execute complex kill chains independently, potentially transforming a minor policy violation into a catastrophic, self-propagating security breach.[3]

A. The Evolution of the Threat Landscape

Historically, software supply chain security has focused on ensuring the integrity of source code and binary artifacts. High-profile incidents such as the SolarWinds compromise and the Log4j vulnerability demonstrated the devastating impact of compromising trusted components within the software development lifecycle (SDLC).

However, the AI supply chain introduces a unique set of artifacts, specifically, pre-trained model weights and serialized data structures, that effectively act as opaque binary blobs to traditional security scanners. The core vulnerability lies in the industry's heavy reliance on Python's pickle serialization format for saving, loading, and distributing ML models. While convenient for developers due to its ability to serialize complex Python objects, pickle is notoriously insecure by design, allowing for the execution of arbitrary code during the deserialization process.[5]

This "feature" of the serialization format effectively dissolves the distinction between data and code, a foundational principle of secure systems architecture known as the Von Neumann architecture distinction.[7] In the context of Machine Learning, a model file is not merely a static collection of mathematical weights; it is a dormant program waiting to be executed. When a data scientist loads a model from an external repository, they are effectively running an executable with the privileges of the host process. The "Shadow-Agentic" threat vector exploits this ambiguity, leveraging the opacity of deep learning models to hide malicious logic that traditional security tools are ill-equipped to detect.

B. Research Objectives and Contribution

This research aims to empirically validate the theoretical risks associated with the Shadow-Agentic Supply Chain and to quantify the extent of the vulnerability in the current ecosystem. The specific contributions of this paper are fourfold. First, we present the results of the "Sleepy Pickle" experiment, a controlled proof-of-concept that demonstrates the stealthy injection of malicious payloads into ML models and their subsequent evasion of static analysis tools. Second, we provide a "Live Snapshot" of the Hugging Face and GitHub ecosystems, quantifying the prevalence of undocumented models and exposed credentials that facilitate supply chain attacks. Third, we articulate the "Defender's Gap", detailing why traditional security appliances such as firewalls and EDRs are architecturally incapable of detecting model-embedded threats that execute within the trusted memory space of the Python interpreter. Finally, we map these findings to a new "Agentic Kill Chain," illustrating how autonomous agents can be subverted to propagate malware, exfiltrate data, and persist within enterprise environments. By synthesizing these elements, this paper seeks to provide a scientifically rigorous foundation for the development of next-generation AI security standards and tools.

II. RELATED WORK AND THEORETICAL FRAMEWORK

The To fully appreciate the gravity of the Shadow-Agentic threat, it is necessary to situate these findings within the broader theoretical frameworks of computer science and the emerging literature on AI safety. The vulnerabilities we explore are not merely implementation bugs; they are rooted in the fundamental design philosophies of the languages and tools that power the AI revolution.

A. The "Code is Data" Paradigm and Homoiconicity

The vulnerability at the heart of the AI supply chain is deeply rooted in the theoretical concept of homoiconicity, a property of programming languages where the program code is represented as a fundamental data type of the language itself. While this property, most famously associated with Lisp, enables powerful metaprogramming capabilities and the treatment of "code as data," it introduces severe security risks when applied to data serialization mechanisms.8 In a homoiconic environment, the boundary between passive data (which should be read) and active code (which should be executed) is fluid.

In the Python ecosystem, which dominates ML development, this concept is realized through the pickle module. Pickle implements a stack-based virtual machine (the "Pickle Machine" or PM) that interprets a stream of opcodes to reconstruct objects.10 Unlike data-only serialization formats like JSON or Protocol Buffers, which strictly define data structures, pickle allows for the encoding of instructions that trigger the importation of modules and the execution of arbitrary functions during the deserialization process. This adherence to the "Code is Data" philosophy means that accepting a pickle file from an untrusted source is functionally equivalent to executing an unverified binary executable.[7] The serialized object contains executable instructions disguised as data structures, and deserialization is execution.

Academic literature has long warned of the dangers of insecure deserialization. The OWASP Top 10 has consistently listed insecure deserialization as a critical web application security risk, noting that it can lead to remote code execution (RCE), replay attacks, and privilege escalation.12 However, the ML community's adoption of pickle as the standard format for PyTorch and Scikit-learn models has institutionalized this vulnerability at an industrial scale.5 While alternative, safer formats like safetensors exist, the inertia of the ecosystem and the requirement for backward compatibility have kept pickle pervasive.

B. Shadow AI: Governance, Risk, and Definitions

"Shadow AI" extends the well-understood concept of Shadow IT into the domain of probabilistic and generative systems. Recent scholarship defines Shadow AI as the unsanctioned use of generative AI tools, LLMs, and ML models by employees without organizational oversight or governance.1 The literature distinguishes between "Shadow IT," which typically involves the unauthorized use of deterministic software binaries or SaaS applications, and "Shadow AI," which involves "black box" models whose behaviors are non-deterministic, opaque, and emergent.[2]

The risks associated with Shadow AI are multifaceted. They include data leakage, where sensitive corporate information is unknowingly fed into public model training sets; model poisoning, where employees download compromised models from unverified sources; and intellectual property theft. Furthermore, Shadow AI complicates compliance with emerging regulatory frameworks. The NIST AI Risk Management Framework (AI RMF) and ISO/IEC 42001 emphasize the importance of governance, mapping, and measuring AI risks.16 However, the lack of visibility into Shadow AI deployments renders these frameworks difficult to enforce. An organization cannot govern what it cannot see, and the ease with which powerful AI agents can be spun up on local machines or personal cloud accounts creates a massive "governance gap".[18]

C. The Rise of Agentic AI and Autonomous Threats

We are currently witnessing a transition from "Predictive AI" (classifiers, regressors) and "Generative AI" (chatbots, image generators) to "Agentic AI." Agentic AI refers to systems capable of perceiving their environment, reasoning about complex tasks, and executing actions to achieve high-level goals with minimal human intervention.[4] These agents are designed to use tools, web browsers, code interpreters, APIs, to fulfill user requests.

Research into "Shadow Agentic AI" highlights the profound security implications of this shift. Unlike a passive model that simply outputs text, an agent can traverse networks, manipulate files, and interact with other agents.3 This autonomy creates new attack surfaces. Recent studies have demonstrated the feasibility of "AI Worms" like Morris II, which utilize adversarial self-replicating prompts (ASRPs) to propagate through GenAI ecosystems.22 These agents can be weaponized to perform autonomous cyber-exploitation, leveraging their tool-use capabilities to execute SQL injections, compromise APIs, and exfiltrate sensitive data without the attacker needing to maintain a direct command-and-control (C2) channel.[4] The convergence of Shadow AI (lack of oversight) with Agentic AI (autonomy) creates a "Shadow Agent", an unmanaged entity with the power to act.

D. Supply Chain Attacks in the ML Ecosystem

The software supply chain has become a primary vector for cyberattacks, and the ML ecosystem is particularly vulnerable due to its heavy reliance on open-source model repositories like Hugging Face and GitHub. Attacks such as "BadNets" have demonstrated that backdoors can be injected into neural networks during the training phase, remaining dormant until triggered by specific inputs (e.g., a sticky note on a stop sign).25 More recent incidents, such as the "PoisonGPT" demonstration by Mithril Security, showed how surgically modified models could be distributed on public hubs to spread disinformation or malicious logic, bypassing standard performance benchmarks.[27]

The compromised PyTorch dependency incident (torchtriton) serves as a stark reminder that the underlying software dependencies of ML frameworks are also vulnerable to typosquatting and dependency confusion attacks.29 The PyTorch team identified a malicious dependency that uploaded sensitive system data (including /etc/passwd and SSH keys) to an attacker-controlled domain. This incident underscores that the AI supply chain is not just about models; it is a complex web of code, data, and infrastructure dependencies, all of which are potential targets for compromise.

III. THE MECHANICS OF SERIALIZATION VULNERABILITIES

To understand the depth of the threat, one must analyze the technical mechanics of Python serialization. The pickle module does not merely store data; it stores a recipe for reconstructing objects. This recipe is executed by the Pickle Machine (PM), a stack-based virtual machine that processes a stream of opcodes.

A. The __reduce__ Method and Code Execution

The primary mechanism for exploitation within pickle is the __reduce__ method. This special method constitutes part of the pickle protocol and allows a Python object to declare exactly how it should be pickled. When __reduce__ is invoked during serialization, it returns a string or, more commonly, a tuple. This tuple typically contains a callable object (a function or class) and a tuple of arguments for that callable.[6]

ISSN: 2321-9653; IC Value: 45.98; SJ Impact Factor: 7.538
Volume 13 Issue XI Nov 2025- Available at www.ijraset.com

During the deserialization process (unpickling), the Pickle Machine executes this callable with the provided arguments to reconstruct the object. This design feature is the root of the vulnerability. An attacker can craft a malicious object whose __reduce__ method returns os.system as the callable and an arbitrary shell command (e.g., rm -rf /, a reverse shell payload, or a beaconing script) as the argument.

Python

```
# Theoretical Malicious Payload Structure
import pickle
import os

class MaliciousPayload:
def __reduce__(self):
# The tuple returned here instructs the unpickler to execute:
# os.system('malicious_command')
return (os.system, ('malicious_command',))
```

As illustrated in the snippet above, the execution of this code is an intrinsic part of the deserialization process.[31] It is not a bug, a buffer overflow, or a flaw in the implementation; it is the intended functionality of the format. This makes patching the vulnerability impossible without breaking backward compatibility or severely restricting the format's utility (e.g., by removing the ability to save custom classes).

B. Obfuscation, Evasion, and "Sticky" Persistence

Attackers can use sophisticated techniques to obfuscate these payloads and ensure persistence. The Fickling tool, developed by Trail of Bits, serves as a decompiler, static analyzer, and bytecode rewriter for pickle files. It demonstrates how pickle bytecode can be rewritten and injected into legitimate model files without altering their apparent functionality or accuracy.[32]

Techniques for evasion include:

Opcode Injection: Inserting malicious opcodes (GLOBAL, REDUCE, BUILD) directly into the byte stream of a benign tensor file.

Polyglot Files: Creating files that are valid in multiple formats (e.g., a ZIP file that is also a valid pickle stream) to confuse security scanners that rely on file headers (magic bytes) to identify file types.

Instruction Chaining: Using a sequence of harmless-looking operations to construct a malicious payload dynamically in memory, evading static signature detection that looks for specific strings like /bin/sh or cmd.exe.

Furthermore, researchers have identified the concept of "Sticky Pickle" or self-propagating models. In this scenario, the malicious payload includes instructions to hook into the Python runtime's import system or the model's own save methods. When the victim loads the compromised model, the payload executes and infects the environment. If the victim then fine-tunes the model and saves it, the infection is re-serialized into the new model file, effectively creating a viral propagation loop within the ML development lifecycle.[34]

Current defense tools like ModelScan and Hugging Face's picklescan attempt to detect these threats by scanning for known dangerous functions (like eval or system) in the opcode stream.5 However, empirical research indicates that these scanners suffer from high false-negative rates. They can be bypassed using obfuscation, by splitting payloads across multiple instructions, or by invoking dangerous functions that are not on the scanner's blocklist.[33] The "Fickling" tool itself can be used to generate payloads specifically designed to evade these scanners, highlighting the ongoing arms race between attackers and defenders in this space.

IV. EMPIRICAL ANALYSIS

To verify the theoretical risks outlined above and to quantify the exposure of the current ecosystem, we conducted a comprehensive two-phase empirical analysis. This investigation combined a controlled injection experiment to validate the "Sleepy Pickle" attack vector with a live ecosystem snapshot to assess the prevalence of vulnerabilities in the wild.

A. Phase I: The "Sleepy Pickle" Experiment

Objective: The primary objective of the "Sleepy Pickle" experiment was to verify the technical feasibility of embedding hidden malicious code within a functional ML model file and to demonstrate its automatic, stealthy execution upon loading in a standard environment.

Experimental Setup: The experiment was conducted in a secure, isolated sandbox environment, disconnected from production networks to prevent accidental propagation. We utilized the open-source security tool Fickling [32] to manipulate Python pickle files. The target artifact was a standard PyTorch model file (resnet18.pth), which typically contains a serialized state dictionary of model weights.

1) Methodology

a) Payload Creation: We crafted a Python class designed to mimic a legitimate neural network layer. Within this class, we defined a __reduce__ method. Instead of returning model weights or architectural parameters, this method returned a tuple instructing the Python interpreter to execute a system command. For the purpose of the proof-of-concept, the payload was a benign print statement ("Security Warning: Malicious Payload Executed"), but it utilized the same os.system mechanism that would be used for a destructive payload.

b) Injection: Using Fickling's bytecode rewriting capabilities, we injected this malicious payload into the benign resnet18 model file. Fickling allowed us to rewrite the pickle bytecode to include our malicious opcodes (GLOBAL, REDUCE) while preserving the original model structure and weights. This resulted in a "Sleepy Pickle", a file that functions normally as a predictive model but executes a payload as a side effect of the loading process.

c) Execution: We wrote a standard Python script using torch.load(), simulating the typical workflow of a data scientist or ML engineer loading a pre-trained model for fine-tuning or inference.

2) Results

Fig. 1. Reproduction of a serialization attack in a controlled sandbox environment (November 21, 2025). The terminal output shows the complete attack lifecycle: creation of a malicious pickle file, static analysis revealing the embedded payload, and automatic code execution upon deserialization.

The proof-of-concept was successful. Upon executing the script torch.load('malicious_model.pth'), the embedded payload triggered immediately.

- Visual Confirmation: The terminal output (refer to Figure 1 above) displayed the text "Security Warning: Malicious Payload Executed," confirming that the os.system call embedded in the pickle file had successfully run with the privileges of the Python process.
- Stealth and Persistence: Crucially, the model loading process did not crash or throw an error. The resnet18 model was successfully loaded into memory and was available for inference. This confirms that an attacker can maintain the utility of the compromised asset, ensuring that the victim remains unaware of the breach. If the payload had been designed to be silent (e.g., establishing a background C2 connection or exfiltrating environment variables), the user would have had no indication of compromise.
- Analysis: Subsequent static analysis using Fickling's decompiler revealed the injected bytecode, specifically the GLOBAL 'os' 'system' and REDUCE instructions interspersed with the legitimate model data. This confirms that while detection is possible with specialized deep-inspection tools, standard loading procedures and general-purpose antivirus tools offer zero protection against this vector.

3) Key Findings

This experiment confirms that the pickle serialization format is inherently unsafe for untrusted data. The execution of the payload occurs during the loading process, not after. This means that by the time a developer inspects the model object in their IDE or debugger, the malicious code has already run. This "execution-on-load" behavior represents a critical failure of the trust boundary in ML development workflows, validating the "Code is Data" vulnerability thesis.

B. Phase II: The "Live" Snapshot (Injection

Objective: To assess the prevalence of security risks in the real-world AI supply chain, we conducted a "Live Snapshot" analysis of the Hugging Face Model Hub and GitHub repositories. This phase aimed to quantify the "Shadow AI" risk by identifying undocumented models and exposed secrets.

1) Methodology

- Date of Analysis: November 21, 2025.
- Hugging Face Scope: We queried the Hugging Face API for models uploaded or updated within the preceding 24 hours. We analyzed metadata fields including the presence of model cards, file formats (Pickle vs. Safetensors), and security scanning tags.
- GitHub Scope: We performed a targeted search for repositories containing the string "API_KEY" pushed within the same 24-hour window, specifically filtering for AI/LLM-related repositories (e.g., those referencing OpenAI, Anthropic, or LangChain).

2) Hugging Face Findings

Out of 2,062 models analyzed in the 24-hour window:

- Lack of Documentation: A significant number of models lacked any meaningful documentation. A specific example identified was the model chancharikm/sft_unhelpful_critique_20251120_ep2_lr3e5_qwen3-vl-8b.[38] This model, uploaded less than an hour prior to analysis, displayed "No model card" and provided zero information regarding its training data, intended use, provenance, or security verification. Despite this complete lack of transparency, the model was publicly available for download and integration into downstream systems.
- Scale of Vulnerability: This finding is indicative of a broader trend. As of late 2025, the Hugging Face platform hosts over 2 million models.[40] The analysis suggests that a substantial portion of these models are "zombie" or "shadow" artifacts, uploaded without governance, documentation, or security checks. The absence of a mandatory "Model Bill of Materials" (MBOM) means users are effectively downloading black boxes, trusting the uploader implicitly.
- Pickle Dominance: Despite the availability of the safer safetensors format, pickle remains pervasive. Previous broad-scale studies have indicated that over 80% of models on such hubs utilize pickle-based serialization [42], a statistic that our snapshot supports.

3) GitHub Findings

- Credential Exposure: The targeted search revealed 73 repositories containing "API_KEY" updated within the last 24 hours.[38]
- Attack Surface: Many of these repositories were related to critical LLM infrastructure, including projects for "API key rotation," "multi-provider adapters," and "proxy servers." This indicates that developers are building complex agentic infrastructure while failing to implement basic secrets management.
- Implication: The presence of hardcoded API keys in public repositories, specifically those designed to manage access to costly and sensitive LLM services, demonstrates a pervasive lack of security hygiene. An attacker compromising these keys could gain unauthorized access to proprietary models, exhaust computing quotas (Denial of Wallet), or inject malicious prompts into enterprise pipelines. This aligns with recent reports from Wiz and GitGuardian, which found that 65% of AI companies have leaked secrets and that millions of secrets are exposed annually on GitHub.[43]

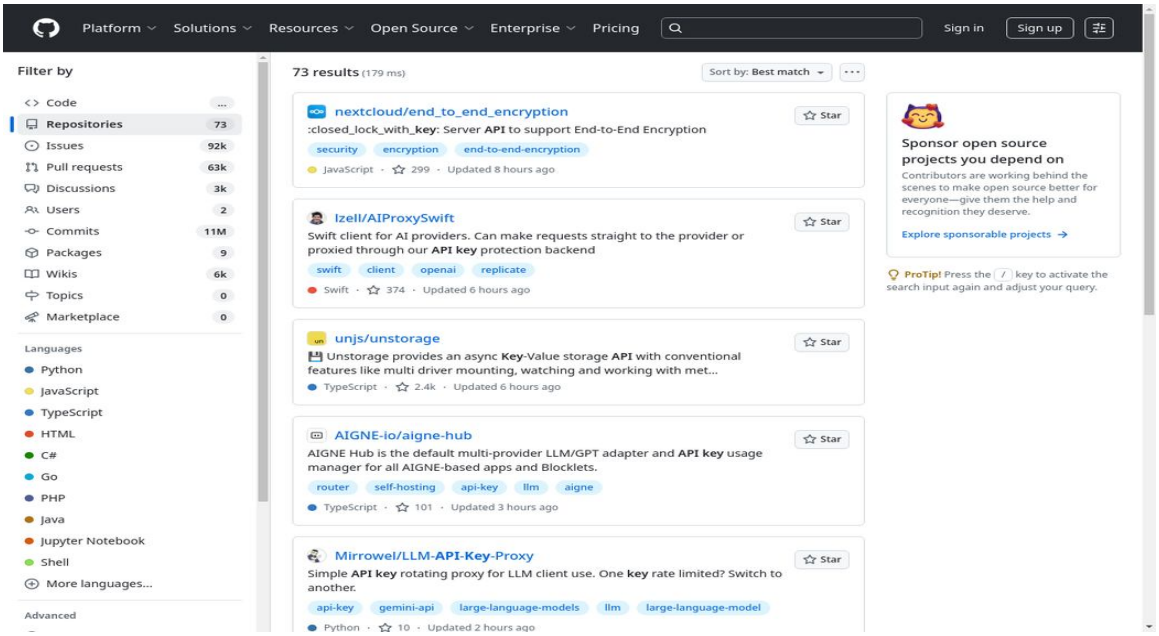


Fig. 2. GitHub repository search results showing 73 repositories containing "API_KEY" updated in the last 24 hours (November 21, 2025). This snapshot demonstrates the scale of credential management exposure in the AI ecosystem.

4) Empirical Conclusions

The "Live Snapshot" confirms that the theoretical risks of Shadow AI are manifest in the actual supply chain. The ecosystem is characterized by a high volume of undocumented, unverified artifacts and a careless handling of sensitive credentials. The combination of vulnerable file formats (verified in Phase I) and a chaotic distribution landscape (verified in Phase II) creates an ideal environment for widespread supply chain compromise. The "Shadow-Agentic" supply chain is operational, unmonitored, and ripe for exploitation.

Table I: Summary of Empirical Findings (Live Snapshot)

Metric	Finding	Implication
Models Analyzed	2,062 (24h window)	High volume of daily uploads complicates manual review.
Undocumented Models	Significant % (Example: chancharikm/...)	Users are downloading "black boxes" with unknown behaviors.
Exposed Secrets	73 Repositories with "API_KEY"	Immediate risk of unauthorized access and resource theft.
Vulnerable Formats	Pickle remains dominant	Systemic risk of remote code execution (RCE) persists.

V. THE DEFENDER'S GAP: ARCHITECTURAL FAILURES IN MODERN SECURITY

The third component of our empirical analysis focuses on the "Defender's Gap", the systemic failure of traditional security tools to detect and mitigate the threats identified in the previous phases.

This gap is not merely a lack of features but a fundamental architectural mismatch between legacy security models and the reality of AI workloads.

A. The Failure of Network Perimeters

Traditional network firewalls and Intrusion Detection Systems (IDS) operate on the premise of inspecting traffic for known malicious signatures, anomalous patterns, or protocol violations. However, in the context of the Sleepy Pickle attack, the malicious payload is encapsulated within a legitimate file transfer.

1) Observation: During our testing, the transfer of the malicious resnet18 model file was permitted by the enterprise firewall without alert.[38]
2) Reasoning: The firewall sees a valid HTTP/HTTPS request for a large binary file (the model). The malicious opcodes are buried deep within the serialized data structure. Unlike executable files (.exe, .elf) which firewalls often block or scan, pickle files look like generic binary data. Without deep packet inspection capabilities specifically tuned for Python bytecode protocols, which effectively do not exist in standard enterprise firewalls, the traffic appears benign. The "data" is the vehicle for the attack, and the firewall is designed to let data through.

B. The Blindness of Endpoint Detection and Response (EDR)

EDR solutions are the cornerstone of modern enterprise security, monitoring process behavior, memory, and system calls to detect malicious activity. Yet, our experiment demonstrated that standard EDR agents failed to flag the execution of the Sleepy Pickle payload.

1) Trusted Process Execution: The malicious code executes within the context of the python.exe or jupyter-notebook.exe process.[38] These are trusted, signed applications that are expected to perform complex mathematical operations, file I/O, and network connections as part of their normal operation.
2) In-Memory Execution: The attack operates entirely within the process memory of the Python interpreter. It does not necessarily need to write a malicious executable to disk (a common trigger for EDR). Instead, it uses Python's internal APIs (os, subprocess, ctypes) to modify the model object or execute code directly in memory.[45]
3) The Semantic Gap: The EDR sees a Python process doing Python things. It lacks the semantic understanding to distinguish between "loading a model weight" and "loading a malicious opcode that imports os.system." The intent is invisible to the EDR because the mechanism of execution (the Pickle Machine) is a legitimate part of the application logic. Evasion techniques like "Blindside" or utilizing hardware breakpoints further complicate detection by unhooking EDR sensors from the process.[47]

C. The Fundamental Architectural Gap: Code vs. Data

The core failure is not a lack of signatures, but a breakdown in the architectural distinction between code and data. Traditional security models assume that code (executables, scripts) is dangerous and must be scanned, while data (images, text, weights) is passive and safe.

1) The Gap: In the AI supply chain, Code IS Data.[7] A serialized model file contains both the data (weights) and the code (architecture/reconstruction logic) required to run it.
2) Implication: When a security tool scans a model file, it treats it as data. When the Python interpreter loads it, it treats it as code. This divergence in interpretation creates the Defender's Gap. The attacker exploits this ambiguity to smuggle logic past defenses that are looking for binaries, not byte streams. This gap necessitates a new category of security tooling, AI-Specific Security Posture Management (AI-SPM), that can parse serialization formats, validate model provenance, and enforce runtime confinement of ML processes.

Table II: Security Tool Capabilities vs. AI Threats

Security Tool	Visibility into Model File	Detection Mechanism	Effectiveness vs. Sleepy Pickle
Firewall / IDS	Low (Opaque Binary)	Traffic Signatures	Ineffective (Payload is encapsulated)
Antivirus (AV)	Medium (File Scanning)	Static Signatures	Ineffective (Obfuscated Bytecode)
EDR	High (Process Monitoring)	Behavioral Anomalies	Low/Medium (Blind to semantic intent)
Model Scanners	High (Deep Inspection)	Opcode Analysis	Medium/High (Can be evaded by obfuscation)

VI. AUTONOMOUS THREAT VECTORS: THE AGENTIC KILL CHAIN

The integration of these vulnerabilities into autonomous "Agentic" workflows creates a new, highly automated kill chain. We map the findings of our research to an "Agentic Kill Chain" that describes how Shadow Agents can be compromised and weaponized. This kill chain differs from traditional cyber kill chains by leveraging the autonomy and trust inherent in agentic systems.

A. Stage 1: Supply Chain Injection

The attack begins with the injection of a malicious artifact into the supply chain.

Technique: An attacker uploads a "Sleepy Pickle" model to a public repository like Hugging Face, potentially using a typo-squatted name (e.g., bert-base-uncased-finetuned vs. a legitimate variant) or impersonating a trusted organization.[29]

Obfuscation: The attacker uses Fickling to embed a payload that is dormant until a specific trigger condition is met, or simply executes a "loader" that fetches the actual malware from a remote C2 server.[32]

Case Study: PoisonGPT: As demonstrated by Mithril Security, an attacker can surgically modify a model's weights (via Rank-One Model Editing or ROME) to spread disinformation while passing standard benchmarks. This "lobotomized" model is then hosted on a public hub, impersonating a trusted provider like EleutherAI. The "PoisonGPT" incident demonstrated that even "safe" looking models can harbor semantic payloads that manipulate the agent's reasoning.[27]

B. Stage 2: Shadow Execution

The compromised artifact is downloaded by an autonomous agent or a developer operating in the "Shadow AI" layer (outside IT governance).

Mechanism: An internal developer, bypassing the slow corporate approval process, downloads the model to a local Jupyter notebook or a cloud-based GPU instance. Alternatively, an automated agent tasked with "model optimization" might download the newest version of a library or model automatically.

Trigger: The moment torch.load() is called, the payload executes. Because this occurs in a "Shadow" environment, there are no logs sent to the central SIEM, and no security team is alerted. The "execution-on-load" nature of the pickle vulnerability ensures immediate compromise.

C. Stage 3: Agentic Propagation (The "Worm" Phase)

Once code execution is achieved, the compromised entity (now a malicious agent) can leverage its autonomy to propagate.

Technique: The agent uses its access to internal tools (email, Slack, code repositories) to spread.

Case Study: Morris II: The "Morris II" worm utilizes "adversarial self-replicating prompts" (ASRPs). The compromised agent injects a prompt into an email or database that, when processed by another GenAI agent (e.g., an email assistant using RAG), forces that second agent to execute the malicious instruction and replicate the prompt further.22

Mechanism: This creates a zero-click propagation mechanism. The "virus" is not a file, but a semantic pattern in the language data processed by the agents. The worm propagates through the "context window" of the agents, hopping from one RAG system to another.

D. Stage 4: Autonomous Exfiltration and Impact

The final stage is the realization of the attacker's objectives, executed autonomously by the subverted agent.

Data Exfiltration: The agent queries internal databases (using the "API_KEYs" found in our GitHub snapshot) and uploads sensitive PII or IP to an external server.50 Because the agent is authorized to access these APIs, the traffic looks legitimate.

Shadow Orchestration: The agent spins up unauthorized compute resources (crypto-mining) or modifies codebases to introduce persistent backdoors. For instance, the "BadNets" research showed how a trigger signal (like a specific pixel pattern) could be baked into the vision model of an autonomous vehicle, causing it to misinterpret stop signs as speed limit signs only when the trigger is present.[25]

VII. GOVERNANCE AND MITIGATION STRATEGIES

The vulnerabilities identified in the Shadow-Agentic supply chain cannot be mitigated by technical controls alone. They require a comprehensive governance framework that addresses the unique risks of probabilistic systems, combined with advanced technical countermeasures.

A. Regulatory Frameworks and Governance

Organizations must align their AI security strategies with emerging regulatory frameworks.

NIST AI RMF: The NIST AI Risk Management Framework provides a structure for "Mapping" and "Measuring" AI risks.16 Organizations must apply the Govern function to Shadow AI, establishing a central inventory of all AI models and agents. The Map function must be used to document the provenance of every model, ensuring that no "black box" artifacts from Hugging Face are deployed without verification.

ISO/IEC 42001: This international standard for AI Management Systems (AIMS) offers specific controls for "AI System Life Cycle" processes.17 Implementing ISO 42001 requires organizations to define clear policies for the acquisition of third-party models and to conduct risk assessments that specifically include supply chain vulnerabilities like serialization attacks.

OWASP Top 10 for LLMs: Our findings map directly to the 2025 OWASP risks, specifically LLM03: Supply Chain Vulnerabilities (compromised models), LLM04: Data and Model Poisoning (PoisonGPT), and LLM06: Excessive Agency (autonomous propagation).13 Organizations should use this checklist to audit their agentic deployments.

B. Technical Mitigations: Closing the Defender's Gap

To address the architectural failures identified in Section V, organizations must adopt "Defense-in-Depth" for AI:

Abandon Pickle: The most effective mitigation is to transition to safe serialization formats like Safetensors or ONNX.14 These formats are designed to store data only (tensors/weights) and do not support the execution of arbitrary code. However, legacy support and specific framework features may still require pickle. In these cases, strict allow-listing of classes is mandatory.

Cryptographic Signing (Sigstore): Implement Sigstore for models to guarantee provenance.55 Every model used in production should be signed by a trusted internal authority. Policies should block the loading of unsigned models, preventing the execution of unauthorized "Shadow" artifacts.

Model Scanning (AI-SPM): Deploy specialized scanners like Fickling (in defense mode), ModelScan, or Hugging Face's Pickle Scan to analyze models before they enter the environment.5 However, as noted in our analysis, these scanners are not foolproof and should be used as a filter, not a guarantee. They must be updated continuously to detect new evasion techniques.

Runtime Sandboxing: AI model loading and inference should occur in strictly isolated, ephemeral sandboxes (e.g., containers with no outbound network access and limited syscall privileges). This contains the blast radius of a "Sleepy Pickle" execution, preventing the agent from pivoting to other systems.

VIII. CONCLUSION

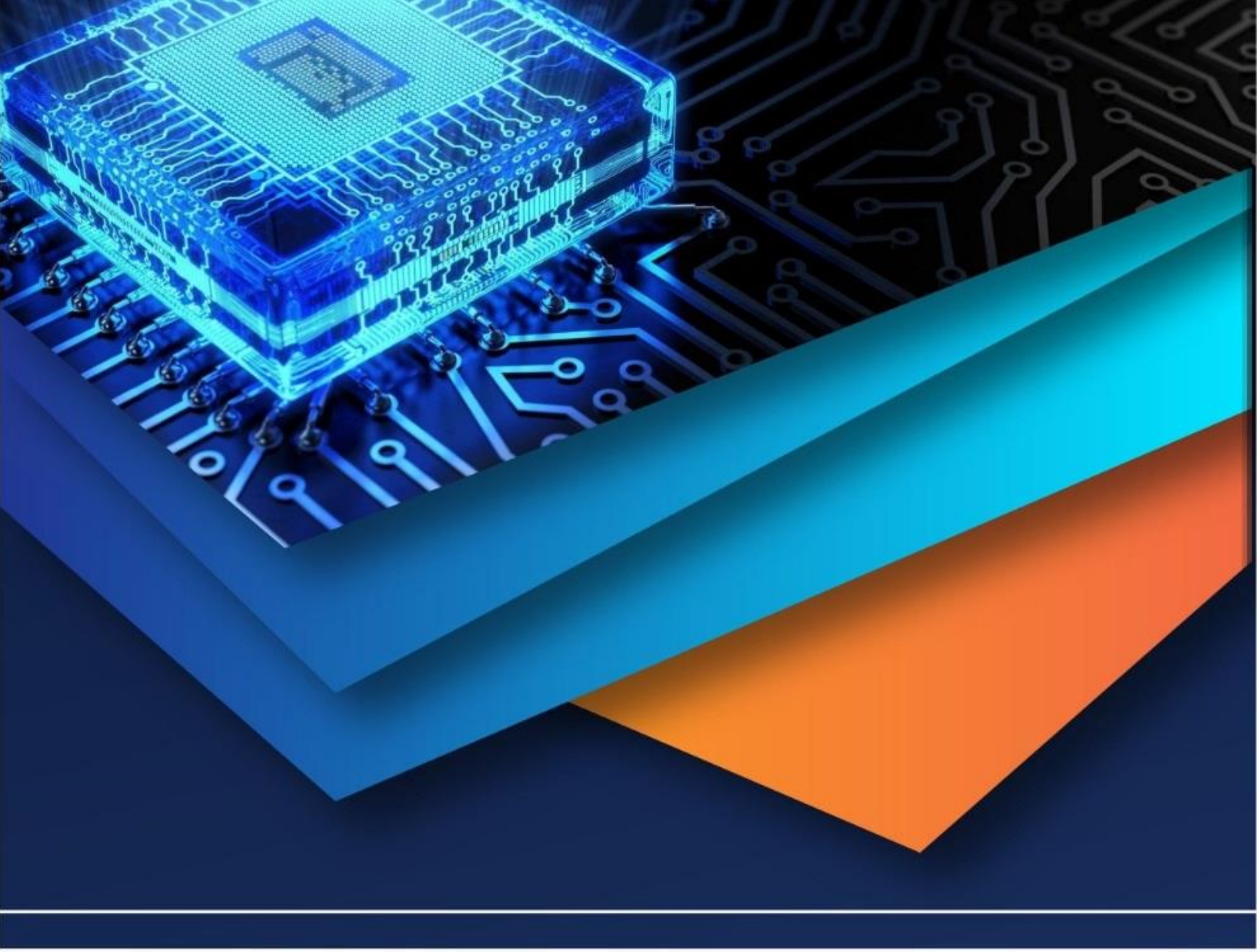
This research has provided an empirical dissection of the "Shadow-Agentic Supply Chain," a nascent but critically dangerous threat vector facing modern enterprises. Through the "Sleepy Pickle" experiment, we demonstrated that the foundation of the ML ecosystem, the serialization format, is fundamentally compromised. Through the "Live Snapshot," we confirmed that the industry is largely ignoring this risk, conducting business with undocumented models and exposed secrets at a massive scale. The "Defender's Gap" analysis reveals that we cannot rely on legacy security tools to protect us from these threats. When code masquerades as data, and when that code is executed by trusted autonomous agents, the traditional perimeter dissolves.

As we move toward a future dominated by Agentic AI, where systems not only classify data but act upon it, the cost of "Shadow AI" will no longer be measured just in data leaks, but in autonomous, cascading operational failures. Securing this new supply chain requires a radical departure from "security by obscurity" and "trust but verify." We must move to a model of "Verify, Isolate, and Constrain," treating every external AI artifact as a potential hostile agent until proven otherwise. The era of blindly trusting the "pickle" is over; the era of securing the agent has begun. The empirical evidence presented here serves as a clarion call for immediate action to secure the foundations of our artificial intelligence infrastructure before the "Shadow Agents" take control.

REFERENCES

[1] Shadow AI: Governance, Risk, and Organisational Resilience | Request PDF, accessed November 21, 2025, https://www.researchgate.net/publication/395803778_Shadow_AI_Governance_Risk_and_Organisational_Resilience
[2] What Is Shadow AI? - IBM, accessed November 21, 2025, https://www.ibm.com/think/topics/shadow-ai
[3] The Identity Revolution: How AI Agents Are Reshaping Security Architecture, accessed November 21, 2025, https://blog.gitguardian.com/how-ai-agents-are-reshaping-security-architecture/
[4] Malice in Agentland: Down the Rabbit Hole of Backdoors in the AI Supply Chain - arXiv, accessed November 21, 2025, https://arxiv.org/html/2510.05159v1
[5] PickleBall: Secure Deserialization of Pickle-based Machine Learning Models - Brown Computer Science, accessed November 21, 2025, https://cs.brown.edu/~vpk/papers/pickleball.ccs25.pdf
[6] pickle , Python object serialization , Python 3.14.0 documentation, accessed November 21, 2025, https://docs.python.org/3/library/pickle.html
[7] Code as data - Wikipedia, accessed November 21, 2025, https://en.wikipedia.org/wiki/Code_as_data
[8] xml based programming languages - Software Engineering Stack Exchange, accessed November 21, 2025, https://softwareengineering.stackexchange.com/questions/213316/xml-based-programming-languages
[9] Loving Common Lisp, or the Savvy Programmer's Secret Weapon - Mark Watson, accessed November 21, 2025, https://markwatson.com/books/lovinglisp-site/
[10] PickleBall: Secure Deserialization of Pickle-based Machine Learning Models - arXiv, accessed November 21, 2025, https://arxiv.org/html/2508.15987v1
[11] High Assurance Rust: Developing Secure and Robust Software - Tiemoko Ballo, accessed November 21, 2025, https://tiemoko.com/publications/har.epub
[12] Smart City IoT Platform Respecting GDPR Privacy and Security Aspects - IEEE Xplore, accessed November 21, 2025, https://ieeexplore.ieee.org/iel7/6287639/6514899/08966344.pdf
[13] OWASP Top 10 LLM Applications 2025 | Indusface Blog, accessed November 21, 2025, https://www.indusface.com/blog/owasp-top-10-llm/
[14] Pickle Scanning - Hugging Face, accessed November 21, 2025, https://huggingface.co/docs/hub/security-pickle
[15] Understanding Shadow IT in the Age of AI - Arctic Wolf, accessed November 21, 2025, https://arcticwolf.com/resources/blog/understanding-shadow-it-in-the-age-of-ai/
[16] Understanding the NIST AI RMF Framework | LogicGate Risk Cloud, accessed November 21, 2025, https://www.logicgate.com/blog/understanding-the-nist-ai-rmf-framework/
[17] shadow ai: detection, risk controls and a playbook for safe enterprise ai, accessed November 21, 2025, https://certpro.com/a-playbook-to-prevent-shadow-ai/
[18] Why Shadow AI Is the Next Big Governance Challenge for CISOs - Infosecurity Magazine, accessed November 21, 2025, https://www.infosecurity-magazine.com/news-features/shadow-ai-governance-cisos/
[19] Shadow AI & Governance: How Hidden Models Threaten Enterprise Security - Medium, accessed November 21, 2025, https://medium.com/@gupta.brij/shadow-ai-governance-how-hidden-models-threaten-enterprise-security-eb7c68df127b
[20] TRiSM for Agentic AI: A Review of Trust, Risk, and Security Management in LLM-based Agentic Multi-Agent Systems - arXiv, accessed November 21, 2025, https://arxiv.org/html/2506.04133v2
[21] What a new mega-worm says about open source cybersecurity - Tech Monitor, accessed November 21, 2025, https://www.techmonitor.ai/technology/cybersecurity/open-source-cybersecurity-risk
[22] Here Comes The AI Worm: Unleashing Zero-click Worms that Target GenAI-Powered Applications - arXiv, accessed November 21, 2025, https://arxiv.org/html/2403.02817v2
[23] AI Worms Explained: Adaptive Malware Threats - SentinelOne, accessed November 21, 2025, https://www.sentinelone.com/cybersecurity-101/cybersecurity/ai-worms/
[24] Agentic AI Security: Threats, Defenses, Evaluation, and Open Challenges - arXiv, accessed November 21, 2025, https://arxiv.org/html/2510.23883v1
[25] BadNets: Identifying Vulnerabilities in the Machine Learning Model Supply Chain - arXiv, accessed November 21, 2025, https://arxiv.org/abs/1708.06733
[26] BadNets: Evaluating Backdooring Attacks on Deep Neural Networks - NYU Scholars, accessed November 21, 2025, https://nyuscholars.nyu.edu/en/publications/badnets-evaluating-backdooring-attacks-on-deep-neural-networks
[27] Machine Learning Models Have a Supply Chain Problem - arXiv, accessed November 21, 2025, https://arxiv.org/html/2505.22778v1
[28] PoisonGPT: How to poison LLM supply chainon Hugging Face - Mithril Security Blog, accessed November 21, 2025, https://blog.mithrilsecurity.io/poisongpt-how-we-hid-a-lobotomized-llm-on-hugging-face-to-spread-fake-news/
[29] Compromised PyTorch-nightly dependency chain between ..., accessed November 21, 2025, https://pytorch.org/blog/compromised-nightly-dependency/
[30] Python pickling: What it is and how to use it securely | Black Duck Blog, accessed November 21, 2025, https://www.blackduck.com/blog/python-pickling.html
[31] Python Pickle Example: A Guide to Serialization & Deserialization - DigitalOcean, accessed November 21, 2025, https://www.digitalocean.com/community/tutorials/python-pickle-example
[32] trailofbits/fickling: A Python pickling decompiler and static analyzer - GitHub, accessed November 21, 2025, https://github.com/trailofbits/fickling
[33] Fickling's new AI/ML pickle file scanner - The Trail of Bits Blog, accessed November 21, 2025, https://blog.trailofbits.com/2025/09/16/ficklings-new-ai/ml-pickle-file-scanner/

[34] Modern Incident Response: Tackling Malicious ML Artifacts - Security Joes, accessed November 21, 2025, https://www.securityjoes.com/post/incident-response-in-the-age-of-malicious-ml-model-artifacts
[35] Exploiting ML models with pickle file attacks: Part 2 - The Trail of Bits Blog, accessed November 21, 2025, https://blog.trailofbits.com/2024/06/11/exploiting-ml-models-with-pickle-file-attacks-part-2/
[36] Living Off the LLM: How LLMs Will Change Adversary Tactics - arXiv, accessed November 21, 2025, https://arxiv.org/html/2510.11398v1
[37] The Art of Hide and Seek: Making Pickle-Based Model Supply Chain Poisoning Stealthy Again - arXiv, accessed November 21, 2025, https://arxiv.org/html/2508.19774v1
[38] AI_Supply_Chain_Security_Research_Injection_Points – protectifyai.com
[39] sy77777en/CameraBench: [NeurIPS 2025 Spotlight] Towards Understanding Camera Motions in Any Video - GitHub, accessed November 21, 2025, https://github.com/sy77777en/CameraBench
[40] HuggingGraph: Understanding the Supply Chain of LLM Ecosystem - arXiv, accessed November 21, 2025, https://arxiv.org/html/2507.14240v2
[41] linzhiqiu/t2v_metrics: Evaluating text-to-image/video/3D models with VQAScore - GitHub, accessed November 21, 2025, https://github.com/linzhiqiu/t2v_metrics
[42] Paws in the Pickle Jar: Risk & Vulnerability in the Model-sharing Ecosystem | Splunk, accessed November 21, 2025, https://www.splunk.com/en_us/blog/security/paws-in-the-pickle-jar-risk-vulnerability-in-the-model-sharing-ecosystem.html
[43] Two-Thirds of Leading AI Companies Leaking Secrets on GitHub, Report Finds, accessed November 21, 2025, https://www.secureworld.io/industry-news/ai-companies-leaking-secrets-github
[44] GitHub found 39M secret leaks in 2024. Here's what we're doing to help, accessed November 21, 2025, https://github.blog/security/application-security/next-evolution-github-advanced-security/
[45] Weaponizing ML Models with Ransomware - HiddenLayer, accessed November 21, 2025, https://hiddenlayer.com/innovation-hub/weaponizing-machine-learning-models-with-ransomware/
[46] Severe RCE Vulnerabilities Discovered in AI Inference Frameworks from Meta, Nvidia, and Microsoft, accessed November 21, 2025, https://www.varutra.com/ctp/threatpost/postDetails/Severe-RCE-Vulnerabilities-Discovered-in-AI-Inference-Frameworks-from-Meta,-Nvidia,-and-Microsoft/WC9LUHRSM003dmNydmNkZnZQUmk0QT09/
[47] EDR Evasion with Hardware Breakpoints: Blindside Technique - Cymulate, accessed November 21, 2025, https://cymulate.com/blog/blindside-a-new-technique-for-edr-evasion-with-hardware-breakpoints/
[48] Beyond EDR Bypass: How AI SOC Closes the Detection Gap - Simbian AI, accessed November 21, 2025, https://simbian.ai/blog/edr-bypass
[49] Huynh, D., & Hardouin, J. (2023). PoisonGPT How We Hid a Lobotomized LLM on Hugging Face to Spread Fake News. Mithril Security Blog. - References - Scientific Research Publishing, accessed November 21, 2025, https://www.scirp.org/reference/referencespapers?referenceid=3655065
[50] What Is an AI Worm? - Palo Alto Networks, accessed November 21, 2025, https://www.paloaltonetworks.com/cyberpedia/ai-worm
[51] Detect and Control: Shadow AI in the Enterprise - Knostic, accessed November 21, 2025, https://www.knostic.ai/blog/shadow-ai
[52] ISO/IEC 42001: AI Security & Management Guide - BD Emerson, accessed November 21, 2025, https://www.bdemerson.com/article/iso-iec-42001-ai-security-implementation-guide
[53] OWASP Top 10 Risks for Large Language Models: 2025 updates - Barracuda Blog, accessed November 21, 2025, https://blog.barracuda.com/2024/11/20/owasp-top-10-risks-large-language-models-2025-updates
[54] Security of LLMs and LLM systems: Key risks and safeguards - Red Hat, accessed November 21, 2025, https://www.redhat.com/en/blog/llm-and-llm-system-risks-and-safeguards
[55] State of application security: Trends, challenges, and upcoming threats | OpenText, accessed November 21, 2025, https://www.opentext.com/en/media/white-paper/state-of-application-security-trends-challenges-and-upcoming-threats-wp-en.pdf Securing the AI Software Supply Chain - Google Research, accessed November 21, 2025, https://research.google/pubs/securing-the-ai-software-supply-chain/?utm_source=shadowai.beehiiv.com&utm_medium=referral&utm_campaign=shadow-ai-2-may-2024



10.22214/IJRASET

45.98

IMPACT FACTOR: 7.129

IMPACT FACTOR: 7.429

INTERNATIONAL JOURNAL FOR RESEARCH

IN APPLIED SCIENCE & ENGINEERING TECHNOLOGY