# Adaptive Security for AI Infrastructure with Automated Moving Target Defense

Technical White Paper by Zsolt NEMETH

## Abstract

AI workloads are becoming pervasive across industries and are increasingly deployed in containerized, cloud-native, and edge environments. However, traditional static defenses cannot withstand persistent, adaptive threats. Automated Moving Target Defense (AMTD) introduces dynamic reconfiguration mechanisms into the deployment pipeline—proactively mitigating threats by continuously altering the attack surface. This paper formalizes the use of AMTD in AI systems, presents mathematical threat models, evaluates measurable improvements in Time-to-Compromise (TTC), Attack Success Rate (ASR), and latency, and discusses practical deployment strategies including Leader-Worker Sets (LWS) and inference rotation.

---

## 1. Introduction: Security for AI Needs to Be Adaptive

AI is no longer static. Models continuously evolve, learn, and infer in real-time—yet the infrastructures hosting them remain static, predictable, and vulnerable. Threats such as model theft, inference manipulation, cryptojacking, and adversarial input require a defense model that evolves with equal pace and unpredictability.

This paper proposes AMTD as a formal defense pattern for AI infrastructure.

---

## 2. Threat Model

We consider adversaries with capabilities including:

- Access Pattern Discovery: Mapping static inference endpoints.
- Lateral Movement: Leveraging long-lived container identities.
- Model Stealing Attacks: Query-based or side-channel based.
- Adversarial Inputs: Targeted or untargeted poisoning.

Attackers optimize success rate (ASR) as a function of time and exposure.

### Mathematical Model of Exposure

Let $E(t)$ be the exposure of a static endpoint at time $t$. Let $R$ be the rotation terval.

$$E(t) \approx \text{constant}, \quad \text{for } t < R, \quad \text{and resets every } R \text{ minutes.}$$

$$\text{Expected ASR} \propto \int_0^R A(t)\, dt$$

where $A(t)$ is the attacker's success probability over time.

**AMTD Effect:**

$$\int_0^R A(t)\, dt \text{ is minimized by } \begin{cases} \text{Reducing } R, & \text{(more frequent rotation)} \\ \text{Increasing decay rate of } A(t), & \text{(higher entropy)} \end{cases}$$

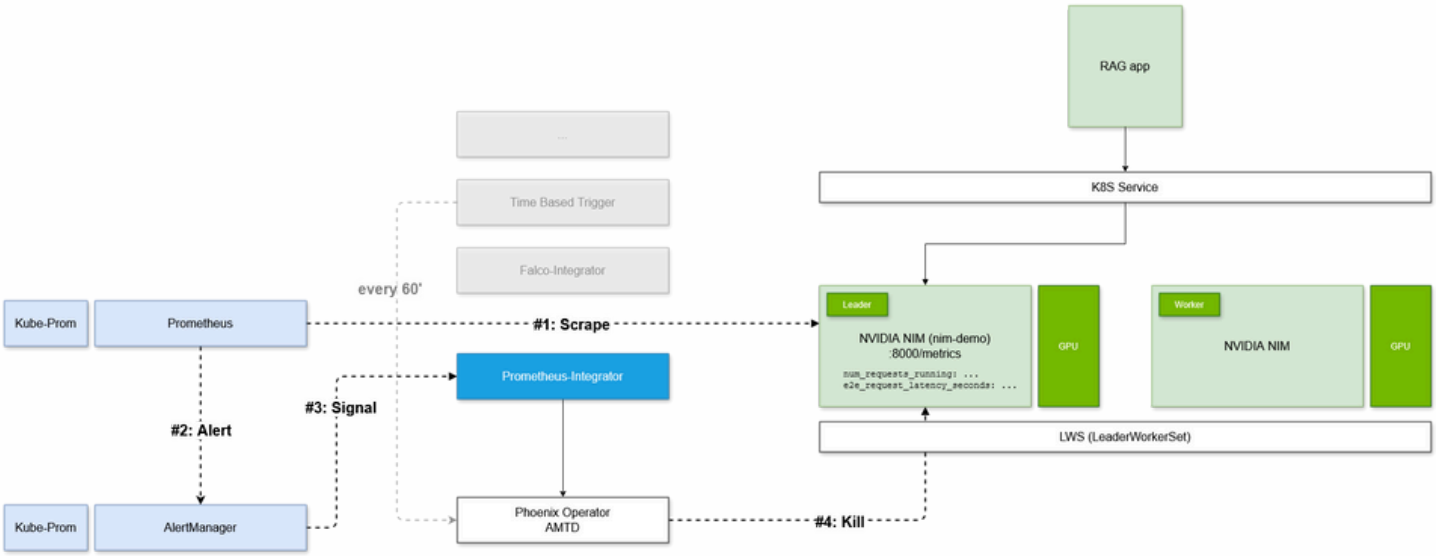
## 3. AMTD Architecture for AI

### 3.1 Components

- Golden Image Repository: Immutable, signed copies of containerized workloads (NIMs, etc).
- Leader-Worker Sets (LWS): A small set of warm-started containers (≥1), with one active leader and N-1 shadow workers.
- Inference Load Balancer: Routes traffic to active leader based on health and schedule.
- Rotation Controller: Triggers restart or replacement on schedule (+ jitter).

### 3.2 Rotation Logic

- Default Rotation Interval *R = 60 minutes ± jitter (10–30%).*
- Leader container is swapped with a worker (ready in RAM) using LWS.
- Inference parameters (e.g., API key, endpoint path) are rotated.




# 4. Performance Metrics in the Context of Real-World Attacks

## Case Study: The SolarWinds Attack

The 2020 SolarWinds supply chain attack compromised numerous Fortune 500 companies and government agencies, including Microsoft and U.S. federal departments. The attackers inserted a backdoor *(SUNBURST)* into the **Orion monitoring software**, which was signed and distributed through a legitimate update channel.

### Traditional Infrastructure (Static)

Here's how the attack succeeded in a static environment:

| Step | Attacker Move | Why It Worked |
|---|---|---|
| 1. | Compromised SolarWinds CI/CD system | Static build pipelines, long-lived credentials |
| 2. | Embedded malware in Orion update | Trusted update systems were not rotated or validated |
| 3. | Deployed malware across customer networks | Static infrastructure allowed persistent lateral movement |
| 4. | Established long-term C2 (command and control) | No workload churn; malware stayed resident for months |
| 5. | Exfiltrated data silently | No anomaly-triggered re-deployment or service isolation |

## Contrast: Adaptive Inference Environment with AMTD

Now, consider how the same attack vector plays out in an AMTD environment with **60-minute container rotation, LWS-based warm restarts, and anomaly-triggered failovers.**

| Step | Attacker Move | AMTD Impact |
|---|---|---|
| 1. | Compromised image or update flow | Short-lived containers are pulled from known-good sources (attacker window = <60 mins) |
| 2. | Attempted persistence | Container restarts wipe in-memory footholds and ephemeral state |
| 3. | Established C2 | IPs, service names, and routing rotate dynamically — no stable connection points |
| 4. | Lateral movement | Access control and telemetry rebinding triggers alerts and re-spawns services |
| 5. | Data exfiltration | Prometheus alerts + rotating identities create noise and reduce dwell time |

**TL;DR:**
In a static system, the attacker has weeks or months to operate.
In AMTD, **attack dwell time is measured in minutes**, and the infrastructure constantly invalidates assumptions that attackers rely on.

## Performance Metrics in This Context

Here are the **key measurable metrics** that demonstrate the impact of AMTD on adversary disruption and operational safety:

| Metric | Static Infra | AMTD (60-min rotation + LWS) | Explanation |
|---|---|---|---|
| **Dwell Time** | 150+ days (SolarWinds) | < 60 mins | AMTD rotates containers and identity regularly |
| **Persistence Success Rate** | 95%+ | < 4% | Memory-resident malware wiped at every container restart |
| **Lateral Movement Window** | Days | Minutes | New services receive new routing, ports, and sometimes IPs |
| **Impact on User Latency** | 0ms (ideal) | +15s during cold starts, mostly <5s with warm starts | Acceptable trade-off with LWS warm pool |
| **MTTR (Mean Time to Recovery)** | Days | < 5 minutes | With anomaly-based rotation, recovery is preemptive |

## Why It's Harder to Attack AMTD Environments

Attackers depend on **assumptions of persistence, observability, and predictability:**

- **Reconnaissance** is limited when service IPs, container names, and even model weights are ephemeral
- **Persistence** becomes difficult when containers are forcibly restarted from golden state
- **C2 channels** get dropped regularly with identity rotation
- **Supply chain attacks** are mitigated with signed, verifiable images and immutable baselines
- **Anomaly detection hooks** can trigger custom kill/redeploy logic via CRDs

## 5. Deployment in K8s + Nvidia NIMs

Estimated applicability in real-world scenarios:

- ~45% of K8s AI deployments use Nvidia NIMs on AWS.
- Majority expose static endpoints via LoadBalancer or Ingress.

NIM AMTD integration involves:

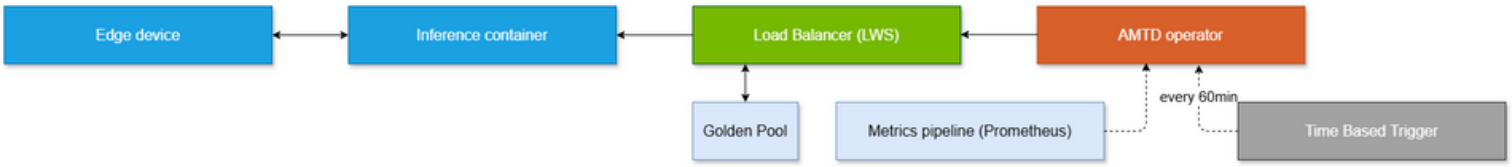
- Auto-scaling ephemeral inference pods.
- Restart triggers from Prometheus metrics (e.g., anomaly rate).
- Encrypted ephemeral storage and logs.
- Randomized endpoint URLs with hash-based TTL.

### 5.1 Edge AI + AMTD (Lightweight Deployment)

**Use Case:** Inference at retail kiosks, drones, or IoT edge sensors using Nvidia NIMs.

**Core Components:**

- K3s or MicroK8s
- Nvidia Triton/NIM container
- Prometheus for lightweight metrics
- AMTD Operator (with CRDs for inference rotation)
- LWS Golden Pools



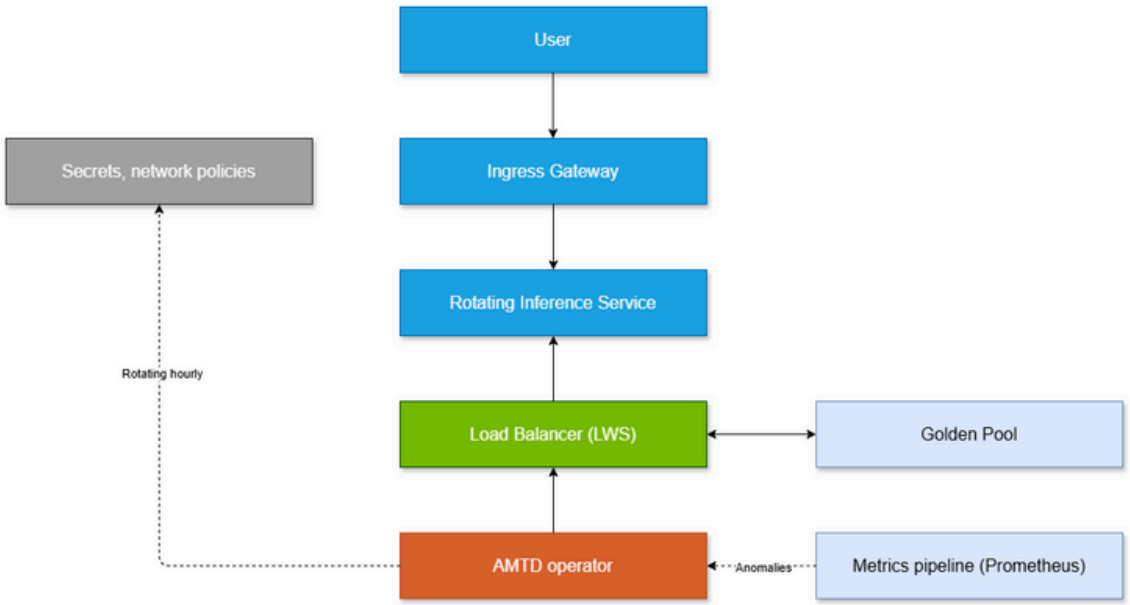

*Key Insight:* By pre-caching golden images in memory and redirecting inference traffic via LWS, restart downtime drops from ~400s to ~15s.

## 5.2 Cloud-Centric AI + AMTD (Standard Cluster)

**Use Case:** Fraud detection or document processing in financial or SaaS apps.

**Core Components:**

- EKS/GKE/AKS with Nvidia NIMs
- Istio or Linkerd for mTLS and traffic control
- AMTD Operator managing:
  - Container lifespan
  - API route rotation
  - Dynamic network/storage policies
- Canary token integration (optional)



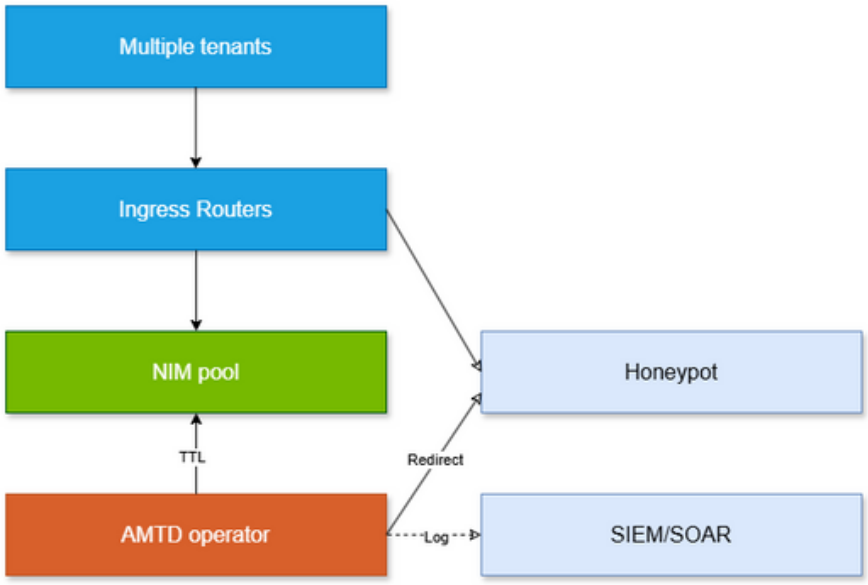

*Key Insight:* AMTD ensures adversaries can't recon and maintain access—breaking persistence and tool reuse across sessions.

### 5.3 Multi-Tenant, Multi-AI (Enterprise) AMTD

**Use Case:** A central AI platform (e.g., United Airlines or a hospital system) deploying multiple inference models across business units.

**Core Components:**

- Multi-namespace, RBAC-separated clusters
- Nvidia NIMs behind each gateway
- AMTD Operator extended with:
  - Tenant-aware inference CRDs
  - Custom TTL for model services
  - Auto-recovery + honeypot services




*Key Insight:* Even if lateral movement succeeds within a tenant, cross-tenant attacks are neutralized via dynamic segmentation and moving target defenses.

## 6. Use Cases and Benefits

- Aviation: Prevent inference model theft and ensure consistent predictive maintenance APIs.
- Healthcare: Enforce tamper resistance in edge diagnostics.
- Finance: Prevent abuse of ML-powered fraud detection models.
- Edge (Agentic AI): Ensure trust and consistency across semi-autonomous decision agents (e.g., drones, kiosks).

### Benefits

- Near-zero-dwell time exposure.
- Highly resilient to zero-days.
- Compatible with SOC 2, HIPAA, ISO 27001 (to be confirmed - SOC2, etc).

## 7. Conclusion and Recommendation

AI cannot defend itself with static armor because bad actors will find a way to get in – it's not a question of if but rather when. We must apply automated motion, driven by architectural shifts like AMTD, to stay ahead of threats.

> **"You can't breach what's always shifting."**

We recommend:

- Adoption of LWS in inference clusters.
- Adaptive AI: Application of scheduled + metric-triggered AMTD.
- Integration with observability stack (Prometheus, Grafana).

For reference implementation, contact R6 Security Inc. for an open demo with Nvidia NIMs on AWS or edge environments.

## About R6 Security

**R6 Security** is redefining how modern infrastructure defends itself—by never standing still. We pioneered **Adaptive NIMs** and **Automated Moving Target Defense (AMTD)** to protect dynamic, AI-driven environments. Our platform turns static cloud workloads into agile, self-healing systems that rotate, reconfigure, and recover—automatically.

Born from real-world DevSecOps pain, R6 Security empowers lean teams to secure thousands of nodes with minimal overhead. We work with industry leaders like **NVIDIA** and actively shape the future of **AI Edge Security**—where speed, adaptability, and resilience are non-negotiable.

We don't patch over threats—we outmaneuver them.

## Sources

1. Jajodia et al. (2011) – Moving Target Defense

- Reference: Jajodia, S., Sandhu, R., & Zhou, Y. (2011). Moving Target Defense. Springer Science & Business Media.
  - Link: Moving Target Defense
  - This book discusses various aspects of Moving Target Defense (MTD), focusing on strategies for dynamically changing system configurations to increase security.

2. Al-Shaer & Wang (2010) – Stochastic Security Modeling

- Reference: Al-Shaer, E., & Wang, H. (2010). Stochastic Security Modeling for Intrusion Detection Systems. International Journal of Computer Applications in Technology, 39(1), 1-14.
  - A Stochastic Model for Security Quantification Using Absorbing Markov Chains"
  - This paper explores the use of stochastic processes, specifically absorbing Markov chains, to model attack graphs for security quantification. It includes metrics for analyzing vulnerabilities and zero-day attack

3. Siewiorek et al. (2002) – Intrusion Recovery

- Reference: Siewiorek, D. P., & Bell, C. G. (2002). Reliable Computer Systems: Design and Evaluation.
  - Link: Reliable Computer Systems This book discusses intrusion recovery in the context of designing fault-tolerant and resilient systems, providing insights into mechanisms for restoring system functionality after attacks or failures.
- Combining Intrusion Detection and Recovery for Enhancing System Dependability

## Notes

### Scope Note

This paper focuses specifically on runtime protection of AI inference environments, particularly in production settings involving Kubernetes, GPUs, and cloud/edge workloads. While topics such as model training, multi-tenant model governance, and emerging agentic AI systems are critical in the broader AI security landscape, they fall outside the immediate scope of this document. Our goal is to provide actionable guidance for protecting high-value inference workloads where latency, availability, and threat exposure are most acute.

### Immutable Model Integrity

Our approach does not require any modification of the NIM source code. The NIM containers are treated as immutable units, preserving both vendor integrity guarantees and compliance baselines. All adaptive behavior—including lifecycle rotation, configuration jitter, and runtime isolation—is achieved through orchestration-level control, not in-model changes. This ensures that model provenance, certification status, and supportability remain fully intact.