Threat Model: Securing ClawsBot/MoltBot

Kaushikq Ravindran

This document defines the threat model for Secure MoltBot Bedrock, an AWS deployment of MoltBot with application-level security enhancements. It identifies assets, threat actors, attack vectors, and mitigations developed in response to the January 2026 MoltBot incident where 1,000+ gateways were exposed due to a localhost bypass vulnerability. The security architecture implements five core modules: PermissionSystem, ActionValidator, RateLimiter, AuditLogger, and BedrockSecurityMiddleware.

AI Security | Threat Modeling | AWS Bedrock | Prompt Injection

This document defines the threat model for Secure MoltBot Bedrock, an AWS deployment of MoltBot with application-level security enhancements. It identifies assets, threat actors, attack vectors, and mitigations. Figure 1 shows the complete security architecture.

Context: On January 23, 2026, the MoltBot incident exposed 1,000+ gateways due to a localhost bypass vulnerability. This project exists to prevent similar incidents in cloud deployments.

1. Problem Statement

The original MoltBot Gateway trusts localhost connections by default. When deployed behind a reverse proxy, all traffic appears local, bypassing authentication entirely.

Risks identified in the incident: Credential theft (API keys for LLMs, databases, cloud services); Agent function hijacking (impersonation, phishing); Arbitrary command execution (full system takeover).

A. Design Principles. Our security enhancements are based on the moltbot-safe design philosophy:
Least Privilege: Agents receive only permissions they need.
Explicit Grants: No default capabilities, all declared.
Isolation: Actions sandboxed from host system.
Transparency: Every action logged (allowed and denied).
Auditability: Logs easy to review and verify.
Safety Over Capability: Security always wins over features.

2. Assets

We protect the following assets:

Significance Statement

On January 23, 2026, the MoltBot incident exposed over 1,000 gateways due to a localhost bypass vulnerability, compromising API keys and enabling arbitrary command execution. This threat model documents the security enhancements implemented in Secure MoltBot Bedrock to prevent similar incidents in AWS cloud deployments.

[1]To whom correspondence should be addressed. E-mail: kaushikq.ravindran@gmail.com

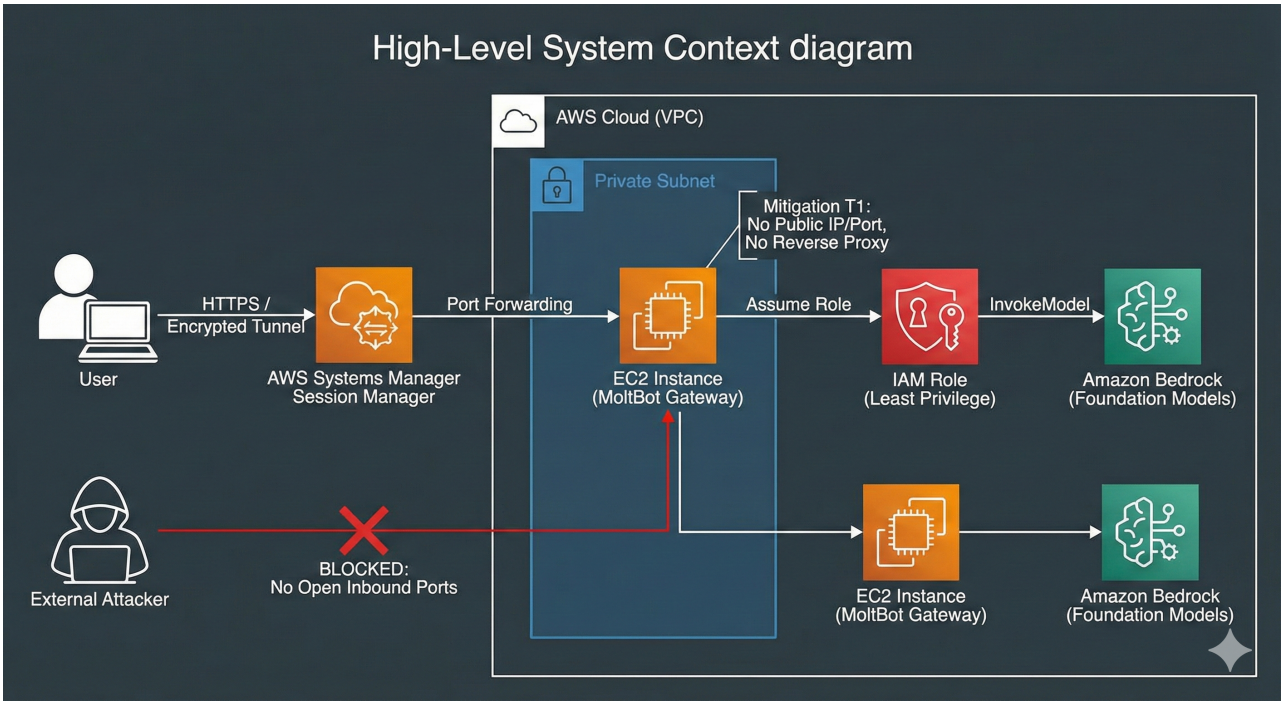

Fig. 1. Security Architecture: User input flows through BedrockSecurityMiddleware (ActionValidator → PermissionSystem → RateLimiter → AuditLogger) before reaching Bedrock API. AWS infrastructure layer provides IAM roles, VPC endpoints, SSM, and CloudTrail.

Bedrock API: IAM role credentials → Cost explosion, data exfiltration.
Gateway Token: Auth token for UI → Full agent control.
Host Filesystem: EC2 instance files → Data theft, malware.
Conversations: Chat history/context → Privacy violation, PII.
Audit Logs: Security records → Cover tracks, compliance.
Permission Config: permissions.json → Privilege escalation.
API Keys: Third-party creds → Lateral movement.

3. Threat Actors

External Attacker (Untrusted): Network access → Steal credentials.
Malicious Agent (Untrusted): Prompt injection → Escalate privileges.
Compromised User (Low trust): Valid credentials → Misuse access.
Buggy LLM (Variable): Hallucinations → Accidental damage.
Misconfigured Admin (Trusted): Full access → Create vulns.

4. Attack Vectors

A. The January 2026 Attack (Localhost Bypass). The gateway trusted `X-Forwarded-For` headers without validation. The attack flow is illustrated in Figure 2.

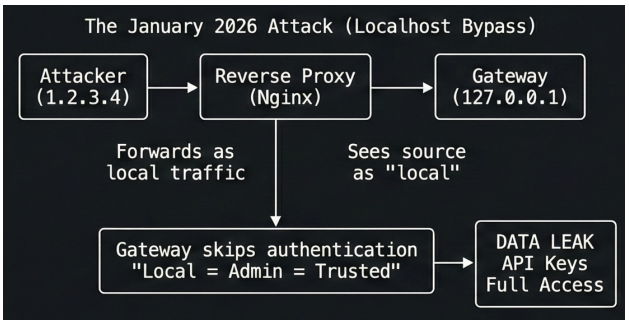

Fig. 2. Localhost Bypass Attack Flow: Attacker sends request through reverse proxy, which forwards to Gateway as local traffic, bypassing authentication.

B. Prompt Injection Attack. Figure 3 shows the comparison between unprotected systems and our mitigation approach using pattern-based detection.

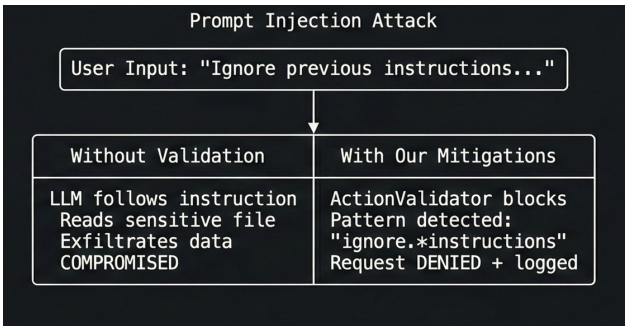

Fig. 3. Prompt Injection Comparison: Without validation, LLM follows malicious instructions. With ActionValidator, patterns are detected and blocked.

C. Privilege Escalation via Model Abuse. Figure 4 shows how the PermissionSystem blocks unauthorized model access.

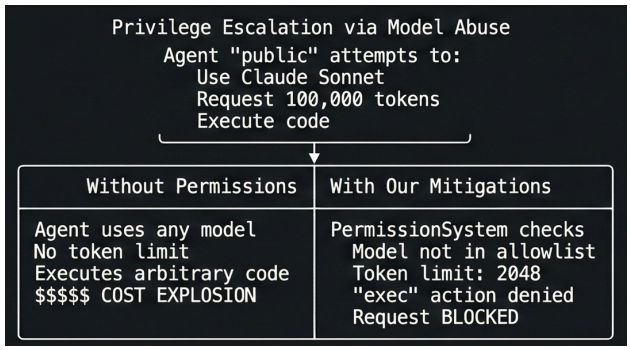

Fig. 4. Privilege Escalation: Agent "public" attempts to use expensive models. PermissionSystem blocks unauthorized actions.

D. Rate Limit Abuse (Cost Attack). Figure 5 demonstrates the cost savings from rate limiting.

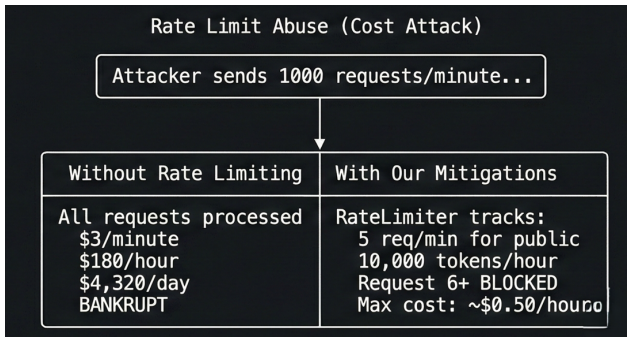

Fig. 5. Rate Limit Abuse: Without rate limiting, 1000 req/min costs $4,320/day. RateLimiter caps at ~$0.50/hour.

5. In-Scope Threats

T1 Localhost bypass (Critical) → SSM Session Manager
T2 Prompt injection (High) → ActionValidator
T3 Model abuse/cost (High) → PermissionSystem
T4 Token limit bypass (Medium) → PermissionSystem
T5 Rate limit abuse (Medium) → RateLimiter
T6 Privilege escalation (High) → Per-agent policies
T7 Audit log tampering (Medium) → Append-only logging
T8 Tool abuse (High) → Tool allow/deny lists
T9 Overly broad IAM (Medium) → Restricted model ARNs
T10 Insecure auth (High) → allowInsecureAuth: false

6. Out-of-Scope Threats

AWS infrastructure attacks (AWS responsibility); OS-level container escapes (Requires Docker/OS hardening); Physical access to EC2 (AWS data center security); Supply chain attacks (Upstream project responsibility); DDoS against AWS (AWS Shield responsibility).

7. Mitigation Matrix

T1: SSM-only access (CloudFormation, yaml)
T2: Pattern matching (ActionValidator, .py)

T3: Model allowlist (PermissionSystem, .py)
T4: Max tokens (PermissionSystem, .py)
T5: Sliding window (RateLimiter, .py)
T6: Agent policies (PermissionSystem, .json)
T7: Append-only (AuditLogger, .py)
T8: Allow/deny lists (Agent config, .json)
T9: Model ARNs (IAM Policy, .yaml)
T10: Secure auth (Gateway config, .json)

8. Residual Risks

New injection patterns (Medium): Update regex patterns regularly.
Zero-day in MoltBot (High): Monitor upstream, apply patches.
IAM role compromise (High): CloudTrail alerts, rotation.
Insider threat (Medium): Audit log review, least privilege.
Cost estimation error (Low): AWS Cost Explorer alerts.

9. Test Scenarios

Injection block: Input "Ignore previous..." → allowed: false, logged.
Model deny: public uses Sonnet → allowed: false.
Rate limit: 20 req/min (limit: 10) → Requests 11-20 blocked.
Token limit: 10K tokens (limit: 2K) → allowed: false.
Tool deny: public calls exec → allowed: false.
Valid request: main chats Nova → allowed: true.

10. Future Hardening

Container sandboxing (High): Docker with limited capabilities.
Signed audit logs (Medium): Cryptographic signatures.
Anomaly detection (Medium): ML-based behavior detection.
Formal verification (Low): TLA+/TLC for critical paths.
Permission audit (Medium): Weekly review of usage.

Conclusion

The Secure MoltBot Bedrock deployment implements comprehensive application-level security controls that address the vulnerabilities exposed in the January 2026 incident. By combining AWS infrastructure security with per-agent permissions, injection detection, rate limiting, and audit logging, this architecture provides defense-in-depth protection for AI agent deployments.

ACKNOWLEDGMENTS. This work was inspired by the moltbot-safe project and developed in response to the January 2026 MoltBot security incident. Thanks to the AWS Bedrock team for their security documentation.

References

[1] Cisco Security Blog, *Personal AI Agents Like MoltBot Are a Security Nightmare*, January 2026.

[2] MoltBot Safe Project, *Threat Model and Design Philosophy*, 2026.

[3] Amazon Web Services, *Security in Amazon Bedrock*, 2026.

[4] OWASP Foundation, *OWASP Top 10 for LLM Applications*, 2025.