WHEN YOUR MODEL IS THE ATTACK SURFACE

Adversarial AI: Defending Machine Learning Models from Attack

Executive briefing — Security architects and ML engineering leaders | ~20 min

ML Models Are a New Kind of Attack Surface

- Traditional AppSec assumes fixed logic; ML models make probabilistic decisions learned from data, so the attack surface includes the data, the training pipeline, and the model's learned behavior — not just code
- Inputs that look benign to a human reviewer can be crafted to manipulate model output in ways static code review will never catch
- Standard controls (WAFs, code scanning, dependency audits) do not detect adversarial inputs or poisoned training data
- Model behavior can be probed and reverse-engineered purely through API access, without any code-level vulnerability
- Security and ML engineering teams need a shared threat model — this deck is meant to establish that common language

Evasion Attacks: Fooling the Model at Inference Time

- Adversarial examples are inputs deliberately perturbed — often imperceptibly to humans — to cause misclassification or bypass detection
- Applies across domains: image classifiers, spam/fraud filters, malware detectors, biometric and content-moderation systems
- Attackers typically need only query access to a model (black-box) to iteratively craft an effective perturbation
- Industry-reported range: research repeatedly shows undefended image classifiers can be pushed to high misclassification rates with small, targeted perturbations — treat this as a general finding, not a specific verified statistic
- Risk is highest wherever a model output gates a security or financial decision (fraud scoring, access control, content filtering)

Data Poisoning: Corrupting the Model Before It Ships

- Poisoning attacks inject manipulated samples into training or fine-tuning data so the model learns a hidden flaw or backdoor
- Especially dangerous with crowdsourced, scraped, or externally sourced training data, and with continual/online learning pipelines
- Backdoor poisoning can implant a trigger pattern that causes targeted misbehavior only when that trigger appears — invisible under normal testing
- Representative example, not a verified case study: a retrained fraud model that quietly learns to wave through transactions containing a specific merchant code inserted by an attacker into training data
- Data provenance and pipeline integrity become as critical as code integrity

Model Extraction and IP Theft via API Querying

- Attackers can reconstruct a functionally similar copy of a proprietary model by systematically querying its API and training a surrogate on the input-output pairs
- Extraction erodes competitive advantage for proprietary models and can also serve as reconnaissance for crafting evasion attacks
- Risk scales with API exposure — high query volume, verbose outputs (confidence scores, probabilities), and lack of rate limiting all make extraction cheaper
- Extraction is difficult to distinguish from legitimate high-volume usage without behavioral analytics
- Applies to internally hosted models exposed to partners or third parties, not just public-facing products

Model Inversion and Membership Inference: Privacy Risk

- Model inversion attempts to reconstruct representative training inputs (e.g., approximate images or records) from model outputs or gradients
- Membership inference determines whether a specific individual's record was part of the training set — a direct privacy and regulatory exposure
- Risk is amplified for models trained on sensitive data: health records, financial history, biometric data, proprietary customer data
- These attacks generally require only prediction access, not access to the model's internals
- Regulatory frameworks increasingly treat training-data leakage as a reportable privacy incident, not just a technical curiosity

Prompt Injection: The LLM-Specific Attack Class

- Prompt injection manipulates an LLM's behavior by embedding instructions inside content the model processes — user input, documents, retrieved web content, tool outputs
- Direct injection targets the user-facing prompt; indirect injection hides instructions in third-party content the model later ingests (a webpage, an email, a file)
- Consequences include data exfiltration, unauthorized tool/API calls, policy bypass, and manipulated downstream decisions in agentic systems
- Risk grows sharply as LLMs are connected to tools, plugins, and autonomous action — the attack surface is anything the model reads
- Representative example, not a verified case study: an LLM-based support agent that follows hidden instructions embedded in a customer-submitted ticket to reveal internal system prompts

Defense Layer 1: Adversarial Training and Robust Architectures

- Adversarial training augments the training set with adversarial examples so the model learns to resist them — improves robustness but typically trades off some clean-data accuracy
- Certified/provable robustness techniques (e.g., randomized smoothing) offer bounded guarantees for specific perturbation types, at added compute cost
- Ensemble and architectural diversity can reduce the transferability of a single crafted attack across models
- No single technique provides complete protection — robustness is a spectrum to be tuned to threat model and risk tolerance, not a one-time fix
- Robustness gains should be re-validated whenever the model, data, or deployment context changes

Defense Layer 2: Input Validation and Runtime Guardrails

- Input sanitization, anomaly detection, and out-of-distribution filtering can catch inputs that look statistically unusual before they reach the model
- For LLMs: instruction-vs-content separation, output filtering, and tool-call allowlisting reduce the blast radius of prompt injection
- Rate limiting, query pattern monitoring, and output obfuscation (rounding confidence scores) raise the cost of extraction and inversion attacks
- Guardrails should be layered — no single filter reliably catches every adversarial variant, and attackers adapt to static rules
- Treat guardrails as a control that needs its own testing and update cycle, not a set-and-forget configuration

Red-Teaming ML Systems Before Production

- Adversarial red-teaming should test the model itself — not just the surrounding application — using evasion, poisoning, extraction, and injection techniques
- Effective programs combine automated adversarial testing tools with human red-teamers who understand both security tradecraft and ML behavior
- Red-team findings should map to a defined risk severity scale so remediation can be prioritized like any other vulnerability class
- Coverage should include the full pipeline: training data sources, fine-tuning process, API surface, and any connected tools or agents
- Red-teaming is not a pre-launch gate alone — it should recur as the model, data, and integrations evolve

Monitoring for Adversarial Activity in Production

- Query pattern analysis can flag the systematic, high-volume probing characteristic of extraction or evasion attempts
- Confidence-score and output-distribution monitoring can surface drift consistent with poisoning or an emerging adversarial pattern
- For LLM systems, logging and reviewing flagged prompts/outputs helps detect injection attempts and policy bypass over time
- Monitoring for adversarial activity is a distinct discipline from general model performance monitoring and should have its own alerting path to security teams
- Incident response playbooks should explicitly include ML-specific scenarios: poisoning discovery, extraction detection, and injection-triggered data exposure

Governance for Model Risk

- Model risk should be tracked in the same governance structure as other enterprise risk — with ownership, a risk register, and defined escalation paths
- Every production model needs a documented threat model covering evasion, poisoning, extraction, inversion, and (for LLMs) prompt injection
- Data provenance, model lineage, and change history should be auditable — you cannot investigate a poisoning incident without knowing what data trained the model
- Third-party and vendor-supplied models require the same adversarial risk assessment as internally built ones before integration
- Governance should define acceptable risk thresholds per use case — a customer-facing fraud model and an internal analytics model warrant different scrutiny

A Practical Hardening Roadmap

- Near term (0–3 months): inventory production and pilot models, classify by exposure and sensitivity, add basic rate limiting and output obfuscation on public APIs
- Mid term (3–6 months): stand up adversarial red-teaming for highest-risk models, add adversarial-activity monitoring and alerting, formalize data provenance controls
- Longer term (6–12 months): integrate adversarial training and robustness testing into the ML development lifecycle, extend governance and risk registers to cover all production models
- Sequence by exposure and blast radius — internet-facing models and agentic LLM systems with tool access should be hardened first
- Treat this as a continuous program, not a project with an end date — the threat landscape and model inventory both keep moving

Organizational Readiness: Who Owns Adversarial Risk

- Adversarial ML risk sits between security and ML engineering, and without an explicit owner it falls through the gap between them
- Security teams generally need upskilling in ML-specific attack classes; ML teams generally need upskilling in adversarial threat modeling
- A shared vocabulary and joint review process for new model launches prevents the two functions from working in isolation
- Executive sponsorship should come from both a security leader and an ML/AI platform leader, not one alone
- Budget for adversarial red-teaming and monitoring tooling should be planned alongside model development cost, not treated as a later add-on

Next Steps and the Ask

- Approve a joint security and ML engineering working group to own adversarial risk, chartered within the next 30 days
- Fund an initial model inventory and exposure assessment across production and near-production ML systems
- Commission an adversarial red-team engagement on the highest-exposure model(s) — public-facing APIs and any agentic LLM system with tool access — as the first proof point
- Add ML-specific scenarios to existing incident response playbooks ahead of the next tabletop exercise
- Revisit this roadmap quarterly with both security and ML leadership as the model inventory and threat landscape evolve