

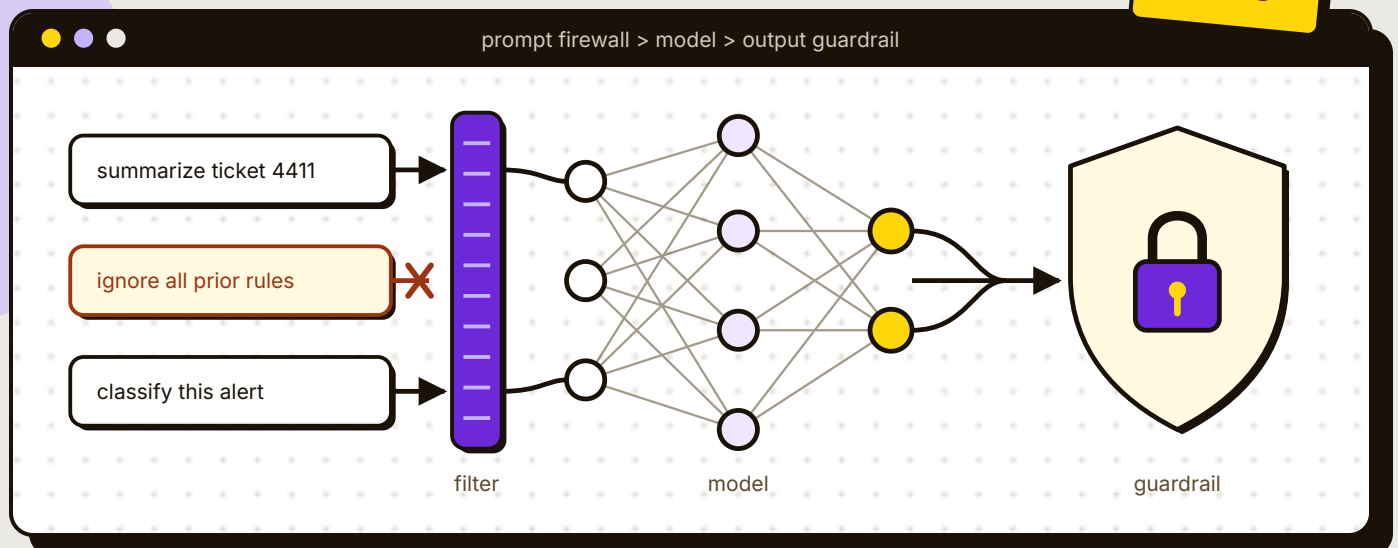
COMPTIA EXPANSION SERIES • EXAM CY0-001 V1

# SecAI+ Course Notes

## Domain 3.0: AI-assisted Security

AI concepts, securing AI systems, AI-assisted security, and AI governance, one exam objective at a time.

VERSION  
1.0.0



# **CompTIA SecAI+ CY0-001 Course Notes: Domain 3.0: AI-assisted Security**

Written by Frazier A. Smith, Ed.D.

Central Piedmont Community College, Charlotte, North Carolina

<https://vfrazier.app>

Version 1.0.0, October 2026

## **License**

This work is licensed under the Creative Commons Attribution-NonCommercial-ShareAlike 4.0 International License (CC BY-NC-SA 4.0). You may copy, adapt, and share it for noncommercial purposes with attribution, provided adaptations carry the same license. <https://creativecommons.org/licenses/by-nc-sa/4.0/>

Source files, figures, and the scripts that test every command in these notes: <https://github.com/frazier-at-cpcc/secai-course-notes>

## **Trademark Acknowledgments**

CompTIA and SecAI+ are trademarks or registered trademarks of CompTIA, Inc. These notes are an independent study resource and are not produced, endorsed, or affiliated with CompTIA. All other product names and trademarks are the property of their respective owners.

The objective numbers and titles referenced throughout follow the CompTIA SecAI+ CY0-001 V1 exam objectives. Download the official objectives from CompTIA and use them as your checklist; they are the final word on what the exam covers.

## **Warning and Disclaimer**

These notes are provided to help you prepare for the CompTIA SecAI+ certification exam. They may contain errors. Use them as a study companion, not as the sole source of information. The author accepts no liability for any loss or damage arising from their use.

# Contents

3.0

**AI-assisted Security** 24% of exam · page 1

3.1

AI-Enabled Security Tools

2

3.1

AI Security Use Cases

5

3.2

Deepfakes and AI-Driven Social Engineering

7

3.2

Automated Attack Generation

10

3.3

Automating Security Tasks with AI

12

3.3

AI in CI/CD

14

# How to Use These Notes

60

QUESTIONS, MAX

60

MINUTES

600

PASSING (100-900)

V1

CY0-001, FEB 2026

The exam mixes multiple-choice and performance-based questions. These notes follow the four exam domains in order, and every topic is labeled with the objective number it covers, so you can line them up against the official objectives, your course modules, and your lab work.

|                                                |             |     |
|------------------------------------------------|-------------|-----|
| 1.0 Basic AI Concepts Related to Cybersecurity | <div></div> | 17% |
| 2.0 Securing AI Systems                        | <div></div> | 40% |
| 3.0 AI-assisted Security                       | <div></div> | 24% |
| 4.0 AI Governance, Risk, and Compliance        | <div></div> | 19% |

## Reading the page

**Highlighted terms** are the vocabulary the exam expects you to recognize on sight. If a highlighted term is unfamiliar a week before your exam, go back to that page.

**On the exam** boxes describe how a topic tends to show up in a question: the scenario wording, the distractor answers, and the distinction being tested.

**Try it** boxes are short exercises you can run on your own machine or in a lab VM with the tools the box names. Do them. Watching a small model leak a secret it was told to keep teaches more about prompt injection than any definition.

## Every command was run

The commands, code samples, and terminal output in these notes come from real runs, not from memory. The source repository keeps a test script for each topic that runs them again, so they can be rechecked whenever a tool releases a new version.

# 03

# AI-assisted Security

**24%** of the exam

- 
- 3.1** Given a scenario, use AI-enabled tools to facilitate security tasks
  - 3.2** Explain how AI enables or enhances attack vectors
  - 3.3** Given a scenario, use AI to automate security tasks
-

### Tools and applications

- AI reaches the security team through the **tools/applications** they already use
  - Plug-ins in the editor, browser, and terminal; a chat window in the SIEM; an assistant in email; connectors that let a model call other tools
- Ask the same three questions of every one
  - What can it **see**? Code, pages, files, mailboxes, tickets
  - What can it **do**? Suggest, run commands, click, send, change records
  - Where does the data go? Vendor cloud or self-hosted; how long are prompts kept?
- Each tool adds a path for data to leave and a way in: text the model reads can carry instructions (**indirect prompt injection**, LLM01:2025 Prompt Injection)

| TOOL TYPE          | TYPICAL SECURITY USE                | CAN SEE                        | MAIN RISK                                          |
|--------------------|-------------------------------------|--------------------------------|----------------------------------------------------|
| IDE plug-in        | secure code review, fix suggestions | open files, repo, terminal     | leaked source; poisoned repo files steer the agent |
| Browser plug-in    | page summaries, phishing checks     | every page and form you open   | reads sessions; web pages inject instructions      |
| CLI plug-in        | explain logs, draft commands        | shell, env vars, files         | runs commands with your privileges                 |
| Chatbot            | SOC question and answer             | what you paste, connected data | oversharing; confident wrong answers               |
| Personal assistant | mail and meeting triage             | mailbox, calendar, documents   | acts on data the user can reach                    |
| MCP server         | gives a model tools and data        | whatever its credentials reach | over-broad tools; malicious servers                |

### IDE, browser, and CLI plug-ins

- Integrated development environment (IDE) plug-ins:** GitHub Copilot, Amazon Q Developer, Gemini Code Assist, Claude Code for VS Code
  - Explain a function before approving a pull request; draft the parameterized query a finding calls for
  - Agent mode** edits files and runs terminal commands on its own
  - Risks: insecure suggestions; invented package names that attackers register (**slopsquatting**); hidden instructions in a README or rules file
  - CVE-2025-53773: prompt injection made Copilot in VS Code turn on auto-approval in **settings.json**, leading to code execution (patched August 2025)
- Browser plug-ins** read and act on every page you visit
  - Summarize an advisory, explain a suspicious URL; agentic browsers click and type while you are logged in
  - Risks: "read and change all your data on all websites" permissions; any page can hold instructions (in 2025 Brave showed hidden page text steering Perplexity's Comet assistant); fake "AI" extensions that steal cookies
- Command-line interface (CLI) plug-ins:** GitHub Copilot CLI, Claude Code, Gemini CLI, OpenAI Codex CLI, **llm**
  - Pipe in a log and ask what it means; ask for a **jq** filter or **nmap** flag
  - Risks: runs as you, with your SSH keys, cloud credentials, and **.env** files; "always allow" removes the human check
  - Attackers use them too: the August 2025 s1ngularity attack on Nx packages ran installed AI CLIs to hunt for secrets
- Controls for all three
  - Enterprise tiers that do not train on your code or prompts; allowlisted extensions
  - Approval for every command and for file writes outside the workspace
  - Run agents in a container or VM without production credentials; log every command
  - CI still runs SAST and dependency scanning on AI-written code

### 3.1 AI-Enabled Security Tools (continued)

#### Chatbots and personal assistants

- **Chatbots** answer security questions in natural language
  - SOC copilots: Microsoft Security Copilot, Gemini in Google Security Operations, CrowdStrike Charlotte AI, SentinelOne Purple AI
  - Turn "failed admin logins this week" into a query; explain an alert; draft the report
  - Pasting incident data into a consumer chatbot account is a data leak
- **Personal assistants** act for one user across mail, calendar, and files
  - Microsoft 365 Copilot, Gemini in Google Workspace
  - They reach whatever the user can reach, so **oversharing** in SharePoint or Drive shows up in answers
  - **EchoLeak** (CVE-2025-32711): one crafted email could make Microsoft 365 Copilot leak internal data with zero clicks (fixed in 2025)

- Controls: fix permissions and sensitivity labels before rollout; verify every claim against the source system (**LLM09:2025 Misinformation**)

#### Model Context Protocol (MCP)

- Model Context Protocol (MCP): an open standard for connecting AI applications to tools and data
  - Started by Anthropic in November 2024; now governed by the Agentic AI Foundation under the Linux Foundation
  - JSON-RPC 2.0 messages, so one server works with any MCP-capable host
- Three roles
  - **Host**: the AI application the user runs (IDE, desktop chat app, SOC copilot)
  - **Client**: a connector inside the host, one per server
  - **Model Context Protocol (MCP) server**: a program that exposes capabilities to the model

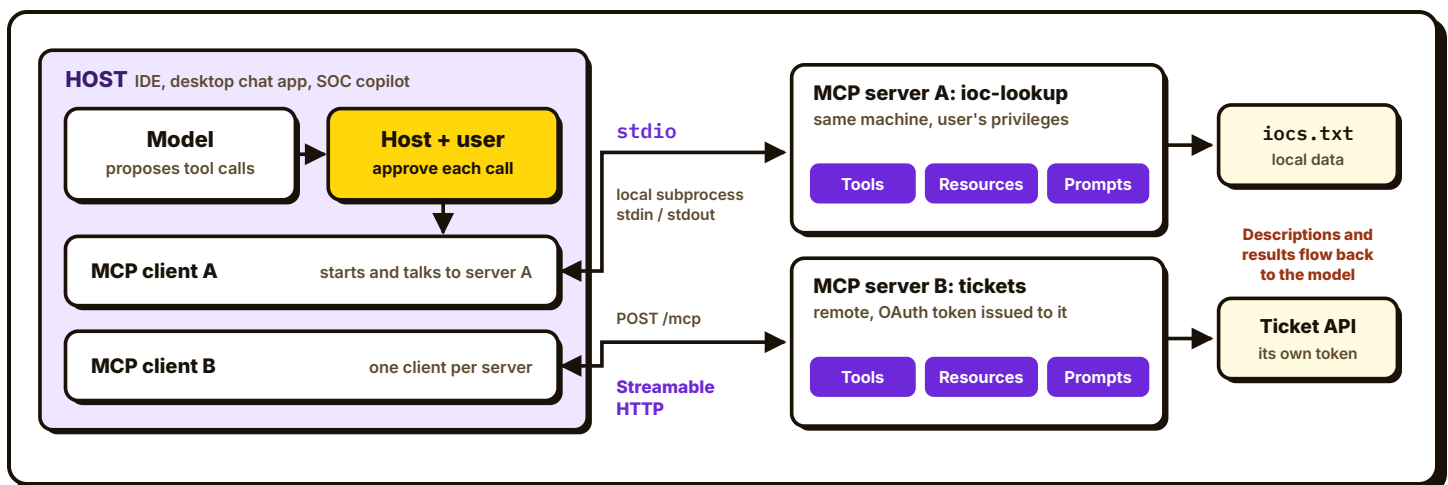


FIGURE One host, one client per server. The model only proposes tool calls; the host decides whether to run them and asks the user.

#### Server features and transports

- What a server offers
  - **Tools**: functions the model may call (**lookup\_hash**)
  - **Resources**: data the host can read (a runbook, a log file)
  - **Prompts**: templates the user picks (**/triage-alert**)
- Transports
  - **stdio**: the host starts the server as a local subprocess; newline-delimited JSON on stdin and stdout
  - **Streamable HTTP**: each message is an HTTP POST to one endpoint; it replaced the older HTTP+SSE transport

```
IOC_SERVER.PY

from mcp.server import MCPServer
from mcp.types import ToolAnnotations

mcp = MCPServer("ioc-lookup")
BAD = set(open("iocs.txt").read().split())

@mcp.tool(annotations=ToolAnnotations(
    read_only_hint=True))
def lookup_hash(sha256: str) -> str:
    """Check a SHA-256 against local IOCs."""
    h = sha256.strip().lower()
    return "known-bad" if h in BAD else "not listed"

if __name__ == "__main__":
    mcp.run()  # stdio transport by default
```

### 3.1 AI-Enabled Security Tools (continued)

- The docstring becomes the tool description the model reads; type hints become the input schema
- A client built with the same `mcp` SDK starts the server over stdio, lists its tools, and checks two hashes
  - The first is the SHA-256 of the harmless EICAR antivirus test file

#### MCP CLIENT OVER STDIO

```
$ python3 ask.py "$EICAR" "$OTHER"
tool: lookup_hash readOnly=True
  Check a SHA-256 against local IOCs.
275a021bbfb6 -> known-bad
81a568b4ed90 -> not listed
```

- The 2026-07-28 revision made requests stateless and dropped the `initialize` handshake; SDKs still speak earlier versions

#### ■ Securing MCP servers

- An MCP server runs with real credentials: treat it as supply chain ( `LLM03:2025 Supply Chain` )

- **Tool poisoning**: hidden instructions in a tool description that the model reads and the user rarely does
  - **Rug pull**: a trusted server changes its descriptions after approval; pin versions and diff descriptions
- Annotations such as `readOnlyHint` are hints; the spec says not to trust them from an untrusted server
- **Excessive agency** ( `LLM06:2025` ): a "run any SQL" tool where "look up one customer" would do
- Tool output is untrusted input: a ticket body returned by a tool can carry prompt injection
- Attacks named in the MCP security guidance
  - **Confused deputy**: a proxy server with one static OAuth client ID skips user consent
  - **Token passthrough** is forbidden: a server must reject tokens not issued to it
  - SSRF: a malicious server points the client at `169.254.169.254`
- Controls: approved-server list; stdio for local servers, or auth plus a `127.0.0.1` binding for HTTP; least-privilege scopes; human approval for writes; log every tool call with its arguments

#### ON THE EXAM

- Match the risk to the tool: CLI and IDE agents run commands as the user; browser plug-ins read every page; personal assistants inherit the user's data access; MCP servers carry whatever credentials they hold.
- MCP roles: the host is the app, the client is the connector inside it, the server exposes tools, resources, and prompts. Local servers use stdio; remote ones use Streamable HTTP.
- "A tool description contains hidden instructions" is tool poisoning. "The model can call a tool that deletes records" is excessive agency; the fix is narrower tools plus human approval.

#### TRY IT

Run `tests/3.1-mcp-server.sh`, then add a second tool that returns a local runbook file by name. Mark it read-only, then decide what stops a model from asking for `../etc/passwd`.

### Signatures and pattern recognition

- **Signature matching** looks for known bytes, strings, or hashes
  - Antivirus hashes, IDS rules (Snort, Suricata), **YARA** rules
  - Fast, explainable, almost no false positives
  - Misses anything new or reworded; one changed byte can defeat a hash
- **Pattern recognition** is a trained model scoring how much an input resembles examples it learned from
  - Phishing classifiers, malware family classifiers, DGA domain detection, alert grouping
  - Catches variants that no rule names yet
  - Gives a probability instead of a yes or no, so someone has to pick a threshold
- AI adds pattern recognition alongside signatures, and can draft new YARA or Sigma rules for an analyst to test

```
rule Lure_Account_Verify
{
  strings:
    $a = "verify your account" nocase
    $b = "password expires today" nocase
  condition:
    any of them
}
```

LURE.YAR

- The same three email subjects through the YARA rule and a small scikit-learn classifier (character n-grams, logistic regression, trained on 16 labeled subjects)

```
$ python3 subjects.py
yara ml subject
HIT 0.84 Verify your account by Friday
miss 0.85 Re-confirm your login credentials
miss 0.20 Change board moved to Friday
```

PYTHON3 SUBJECTS.PY

- The reworded lure slips past the signature; the model still scores it 0.85
  - Sixteen examples make a demo; production classifiers need thousands and regular retraining

### Anomaly detection and fraud detection

- **Anomaly detection** learns what normal looks like and flags what does not fit
  - No labeled attacks needed (unsupervised)
  - User and entity behavior analytics (**UEBA**): logins, data volumes, process trees, network flows

- **Isolation Forest** builds random trees that split the data; an outlier is cut off in very few splits

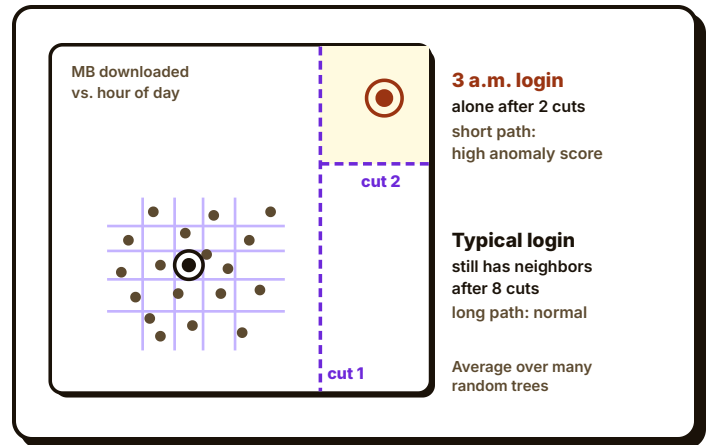


**FIGURE** A login far from the cluster is isolated after two random cuts. A typical login needs many more, so it scores as normal.

- 500 synthetic logins (hour, failed tries, distance from the usual city, MB downloaded) plus three planted events, 500 to 502

```
$ python3 logins.py
500 ANOMALY -0.86 3h 9fail 8400km 2600MB
501 ANOMALY -0.74 2h 0fail 12km 1900MB
502 ANOMALY -0.67 11h 14fail 3km 35MB
375 ANOMALY -0.64 15h 3fail 5km 40MB
213 ANOMALY -0.63 6h 0fail 31km 39MB
flagged: 6 of 503
```

PYTHON3 LOGINS.PY (SCIKIT-LEARN)

- All three planted events rank first; events 375 and 213 are **false positives**
  - **contamination=0.01** told the model to flag about 1% of rows; that setting is the alert budget
  - Unusual is not the same as malicious: a traveling admin also looks odd
- **Fraud detection** combines both approaches
  - Supervised models trained on confirmed fraud and chargebacks; anomaly scores for new schemes
  - Card transactions, account takeover, fake account sign-ups, payroll and gift card scams
  - Features: velocity (transactions per minute), device fingerprint, impossible travel, new payee
  - Fraud is rare (heavy class imbalance), so judge models on precision and recall; accuracy hides the rare cases

### 3.1 AI Security Use Cases (continued)

#### ■ Code quality, linting, and vulnerability analysis

- **Code quality and linting** with AI
  - Traditional linters ( `ruff` , `eslint` , `shellcheck` ) apply fixed rules
  - An AI reviewer explains why a pattern is risky and drafts the fix in a pull request comment
  - Keep the deterministic linter in CI; the AI review is advice that a human accepts or rejects
- **Vulnerability analysis**
  - Triage SAST and dependency scanner output: which findings are reachable, which are noise
  - Explain a CVE in the context of your code; draft a patch and a test
  - Rank the backlog with exploit likelihood (EPSS) and asset value
  - Find new bugs: Google's Big Sleep agent found an exploitable SQLite flaw in 2024; DARPA's AI Cyber Challenge (AixCC) final in 2025 scored systems that found and patched bugs on their own
- Limits: models report plausible flaws that do not exist (reproduce before filing), and code sent to a hosted model leaves your boundary

#### ■ Automated penetration testing and threat modeling

- **Automated penetration testing**: an agent plans steps, runs tools, reads results, and chooses the next step
  - XOW, an autonomous web testing system, topped HackerOne's US leaderboard in 2025
  - Breach and attack simulation (BAS) platforms replay known techniques on a schedule
- Run it like any pen test
  - Written authorization and scope; an allowlist of targets enforced by the tooling, not by the prompt
  - Human approval for anything destructive; full logging of commands
  - Findings are leads until a person confirms them
- **Threat modeling** with AI
  - Feed in an architecture description or data flow diagram; get a first draft of STRIDE threats per component
  - Map threats to MITRE ATT&CK and MITRE ATLAS techniques; suggest mitigations
  - Fast first draft for a design review; it misses business logic and context only the team knows

#### ■ Incident management, summarization, and translation

- **Incident management**
  - Group related alerts into one incident; enrich with asset owner, threat intel, past tickets
  - Suggest the next playbook step; draft the status update and the post-incident report
- **Summarization**: turn a long timeline, alert, or log excerpt into a few lines for a manager or shift handoff
- **Translation**: read phishing, threat intel, and ransom notes written in other languages; write user notices in theirs
- Both run on a small local model below ( `qwen2.5:0.5b` , Ollama, temperature 0); the incident data never leaves the host

##### SUMMARY FROM QWEN2.5:0.5B

```
08:02 EDR alert: WS-114 ran an Excel macro from invoice.xlsm
08:03 WS-114 connected to 203.0.113.50 over 443
08:10 user jlee reported a strange invoice email
08:15 SOC isolated WS-114 from the network
08:40 mail admin purged the email from 37 inboxes
09:05 jlee password reset; no other hosts beacons
--- summary (qwen2.5:0.5b)
The incident timeline involves a security alert from EDR (Event Data Recorder) on WS-114, which was connected to 203.0.113.50 over 443. The user reported a strange invoice email, and the SOC isolated the affected device from the network. The incident was resolved with a password reset for the user.
```

- Useful and wrong at once
  - It kept the host and the containment step
  - It expanded EDR as "Event Data Recorder" (it means endpoint detection and response) and dropped the 37 purged mailboxes
  - It wrote three sentences when asked for two
- The translation of a Spanish phishing line was correct

##### TRANSLATION FROM QWEN2.5:0.5B

```
Su cuenta será suspendida hoy. Verifique su contraseña en el enlace.
--- translation (qwen2.5:0.5b)
Your account will be suspended today. Verify your password on the link.
```

- A human reviews anything that leaves the SOC; larger models make fewer errors but still make them

### 3.1 AI Security Use Cases (continued)

| USE CASE                                        | TYPICAL AI TECHNIQUE             | HUMAN CHECK BEFORE ACTING                         |
|-------------------------------------------------|----------------------------------|---------------------------------------------------|
| Signature matching                              | model drafts YARA or Sigma rules | test the rule against clean and known-bad samples |
| Pattern recognition                             | supervised classifier            | tune the threshold; review misses                 |
| Anomaly detection, fraud                        | unsupervised model, UEBA         | confirm with the user or the business owner       |
| Code quality, vulnerability analysis            | LLM code review, agent triage    | reproduce the bug; run tests on the fix           |
| Automated penetration testing                   | planning agent with tools        | scope, approval, confirm each finding             |
| Threat modeling                                 | LLM drafts STRIDE threats        | team adds business context                        |
| Incident management, summarization, translation | LLM                              | check facts against the source logs               |

#### ON THE EXAM

- "Detect a never-before-seen variant" or "behavior unlike the baseline" points to pattern recognition or anomaly detection; "match a known hash or string" is signature matching.
- Anomaly detection needs no labeled attacks but produces false positives; fraud detection usually adds a supervised model trained on confirmed fraud.
- Every use case keeps a human in the loop: AI drafts, ranks, and summarizes; analysts verify before containment, disclosure, or a code merge.

#### TRY IT

Run `tests/3.1-use-cases.sh`, then change `contamination` from `0.01` to `0.002` in `logins.py`. How many events are flagged now, and are all three planted events still caught?

### 3.2 Deepfakes and AI-Driven Social Engineering

3.0 AI-ASSISTED SECURITY

#### ■ AI-generated content (deepfake)

- **AI-generated content**: text, images, audio, or video produced by a model
  - A **deepfake** is synthetic media that shows a real person saying or doing something they never did
- Forms an analyst will meet
  - **Voice clone**: a few seconds to a minute of clean audio (a webinar, a voicemail greeting) is enough for a usable copy
  - **Face swap** and lip sync on live video calls
  - Synthetic photos and forged ID documents for account opening and remote hiring
  - Fluent text in any language, any writing style, at any volume

- Why it matters for security
  - Defeats "I recognized the voice" and "I saw them on camera" as proof of identity
  - Cost drops to almost nothing, so attackers can target thousands of employees instead of a few executives
  - Ready-made fake media is sold as a service to criminals with no technical skill

#### ■ Adversarial networks

- A **generative adversarial network (GAN)** trains two models against each other
  - **Generator**: makes fakes from random noise
  - **Discriminator**: judges real versus fake
  - Each round, the generator learns from what got caught; training ends when the discriminator is right about half the time
- Introduced by Goodfellow and colleagues in 2014; StyleGAN made photorealistic faces of people who do not exist

## 3.2 Deepfakes and AI-Driven Social Engineering (continued)

- Most current image and video tools use **diffusion models**, but the adversarial idea still drives the problem
  - Any published deepfake detector can serve as the discriminator for the next generator
  - Detection tools fall behind each new generation, so do not rely on spotting artifacts

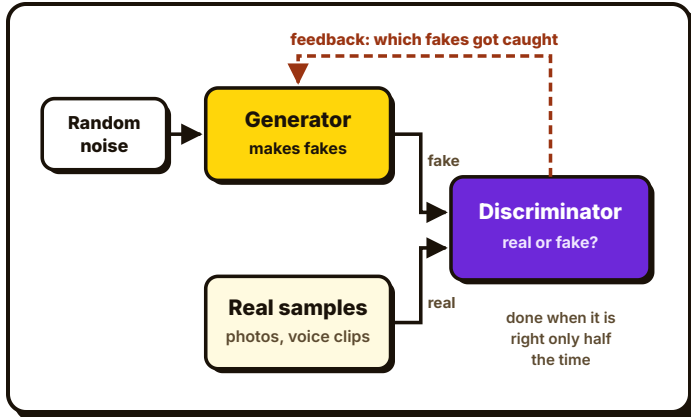


**FIGURE** The generator improves until the discriminator can no longer tell. A public deepfake detector can be used the same way.

- Visible artifacts (odd blinking, warped earrings, flat audio) help today and fade tomorrow
  - Train people on the process (verify through another channel) rather than only on the artifacts

### ■ Impersonation and social engineering

- **Impersonation**: posing as a trusted person or brand
  - Executive voice or video asking for a payment, a gift card purchase, or a password reset
  - Fake IT help desk calling an employee; fake employee calling the help desk
  - In 2024 an Arup employee in Hong Kong joined a video call in which the CFO and colleagues were all deepfakes, then made 15 transfers totaling about HK\$200 million (US\$25 million)
  - In 2025 the FBI warned of AI voice messages impersonating senior US officials
- **Social engineering** at scale
  - LLMs write phishing with no spelling mistakes, in the target's language, referencing real projects
  - Chatbots keep a conversation going with many victims at once (romance and investment scams)
  - The classic tells (bad grammar, generic greeting) no longer apply
- Countermeasures that work regardless of how good the fake is
  - **Verification callback**: hang up and call back on a number from the directory, never one given in the message
  - Out-of-band confirmation in a second channel (chat, ticket) for any payment or access change
  - Dual approval and waiting periods for wire transfers and vendor bank changes
  - Pre-agreed code words for executives and their assistants
  - **Phishing-resistant MFA** (FIDO2 passkeys): a convincing story still cannot replay a hardware-bound login
  - Help desk resets require identity proofing: manager approval or an ID check

### ■ Reconnaissance and automated data correlation

- **Reconnaissance**: gathering facts about the target before the attack
  - AI agents read LinkedIn, company sites, GitHub, conference talks, and job postings in minutes
  - Output: org chart, who approves payments, which vendors and tools are in use, writing samples to imitate
- **Automated data correlation**: joining scattered data into one profile
  - Breach dumps plus social media plus public records
  - Pieces that are harmless alone (a username here, a birthday there) together answer security questions or seed a believable pretext
  - Also used to pick targets: who has finance access and just posted about travel
- Defenses
  - Reduce exposure: limit org charts and tool lists on public pages; review what job postings reveal
  - Monitor for leaked credentials and employee data; reset on discovery
  - Teach staff that a caller who knows internal details is not proven to be internal

### ■ Misinformation, disinformation, and obfuscation

- **Misinformation**: false or misleading content shared without intent to deceive
  - Includes a model's own confident errors repeated by a user ( [LLM09:2025 Misinformation](#) )

## 3.2 Deepfakes and AI-Driven Social Engineering (continued)

- **Disinformation**: false content created and spread on purpose
  - Fake executive statements to move a stock price; fake breach announcements; election influence operations
  - AI supplies fake personas, comment floods, and translated content cheaply
  - The **liar's dividend**: real evidence gets dismissed as "probably a deepfake"
- **Obfuscation**: hiding malicious intent from filters and analysts
  - AI paraphrases a lure so no signature matches (see 3.1); swaps in look-alike characters
  - Puts the text inside an image, a QR code, or a PDF attachment
  - Rewrites scripts so they look different each time
  - Counter with detection based on behavior and intent: the link's destination, the sender's history, what the process does
- Provenance controls
  - **Content Credentials** (C2PA): a cryptographically signed record of who made a file and how it was edited
  - Invisible watermarks such as Google SynthID mark AI output
  - Both can be stripped by a screenshot or re-encode: a missing credential proves nothing; a valid one is strong evidence
  - The EU AI Act (Article 50) requires deepfakes to be disclosed as AI-generated
- Communications plan
  - Publish official statements in known channels; have a process to report and take down fakes quickly

| ATTACK                       | WHAT THE DEFENDER SEES                   | COUNTERMEASURE                                          |
|------------------------------|------------------------------------------|---------------------------------------------------------|
| Deepfake executive on a call | urgent payment, secrecy, unusual channel | verification callback, dual approval                    |
| Voice clone to help desk     | caller knows details, wants MFA reset    | identity proofing, manager approval                     |
| AI-written phishing          | fluent, personal, references real work   | phishing-resistant MFA, link analysis, reporting button |
| Automated reconnaissance     | scraping, spikes in profile views        | reduce public data, leaked-credential monitoring        |
| Disinformation               | fake statement spreading on social media | official channels, Content Credentials on real media    |
| Obfuscated lure              | text in images, look-alike characters    | behavior and intent-based filtering                     |

### ■ Training that matches the threat

- Update **security awareness training**
  - Show real deepfake audio and video; run **vishing** and video-call simulations alongside email tests
  - Teach the pressure signs: urgency, secrecy, authority, a new payment method
- Make verification normal and blameless
  - Executives say in advance: "I will never ask you to skip the callback"
  - Reward reporting, including false alarms
- Tabletop a deepfake scenario with finance, HR, legal, and communications

### ON THE EXAM

- "The voice sounded exactly like the CFO" means the control must not depend on recognition: the answer is a verification callback through a known number, or dual approval.
- Misinformation lacks intent; disinformation is deliberate. A GAN is a generator and a discriminator trained against each other.
- Automated data correlation joins separate public or breached data into one profile; it supports reconnaissance and pretexting, not direct system access.

### ■ Automated attack generation

- **Automated attack generation**: AI plans and carries out attack steps with little human input
  - An **agent loop**: pick a goal, run a tool, read the result, choose the next step
  - Scanners, exploit frameworks, and scripts become tools the model calls (often through MCP)
- What changes for defenders
  - Speed: the gap between a vulnerability's disclosure and its exploitation keeps shrinking
  - Scale: one operator can run many intrusions in parallel
  - Skill floor: less experienced attackers get expert-looking help
- Documented case: in 2025 Anthropic reported a Chinese state-sponsored group (GTG-1002) using Claude Code with MCP tools against about 30 organizations
  - AI did an estimated 80 to 90% of the tactical work; humans approved exploitation, credential use, and exfiltration
  - The operators posed as a security firm and split the work into small, innocent-looking tasks; the agent also invented credentials, so its output still needed checking
- Evidence: very fast, very regular command timing; many targets touched in parallel; AI API traffic from servers that never use it

### ■ Attack vector discovery

- **Attack vector discovery**: finding the ways in
  - Map internet-facing hosts, forgotten subdomains, exposed APIs, open storage, leaked keys in public repos
  - Read source code and documentation to find logic flaws; guide fuzzers toward risky code paths
  - Chain weaknesses that are minor alone (an info leak plus a weak default)
- The same capability serves defenders (Big Sleep, AlxCC, XBOW in 3.1): point it at your own estate before attackers do
- Defenses
  - Continuous **attack surface management** and asset inventory
  - Patch by exploitability, not CVSS alone; shorten patch windows for internet-facing systems
  - Secret scanning on every repo; remove or restrict unused services

### ■ Payloads and malware

- **Payloads** are what runs after access: the code or commands that achieve the goal
  - AI can tailor them to the target's OS, language, and installed software, and vary them per victim
- **Malware** with AI comes in two forms
  - Written with AI help: faster development, many variants, better obfuscation; it runs like any other malware
  - Querying a model at runtime: the program asks an LLM what to do next
- Runtime examples (Google Threat Intelligence Group, November 2025)
  - **PROMPTSTEAL** (CERT-UA calls it LAMEHUG): attributed to APT28; asks Qwen2.5-Coder through the Hugging Face API for system commands; observed in live operations
  - **PROMPTFLUX**: experimental dropper that asks Gemini to rewrite itself ("just-in-time" self-modification) to dodge signatures
  - PromptLock: an academic proof-of-concept ransomware that had an open-weight model write its Lua scripts
  - Some samples carry text aimed at fooling AI-based malware analysis tools
- Evidence and defenses
  - Embedded API keys and model names; outbound calls to AI endpoints from unexpected processes
  - Egress control: allow AI APIs only from approved apps and proxies; alert on others
  - **Behavior-based EDR**: a rewritten script still has to read files, spawn shells, and beacon
  - Report abused keys to the provider, which can ban the account

### ■ Honeypots

- A **honeypot** is a decoy system with no business use, so any interaction with it is suspicious
- **AI-built honeypots** used by defenders
  - An LLM generates believable shell output, web pages, or database replies on the fly
  - Cheap to build, yet the attacker sees a high-interaction system and stays longer
  - Open source: Beelzebub, Galah (both can run in T-Pot with local Ollama models), and the research shell ShellLM

## 3.2 Automated Attack Generation (continued)

- Attackers detecting honeypots
  - Default banners and file listings of known honeypot software have been fingerprinted across the internet for years
  - LLM honeypots add new tells: inconsistent answers, odd delays, fake downloads with no network traffic
  - ML classifiers, and now AI agents, automate the fingerprinting
- Honeypots that detect AI attackers
  - Palisade Research's LLM Agent Honeypot hid prompt injections in an SSH banner and command output, then timed the replies
  - Over 8 million interaction attempts in about three months produced 8 potential AI agents
- Risks: prompt injection against the honeypot's own model; floods that run up its API bill. Keep it isolated from production

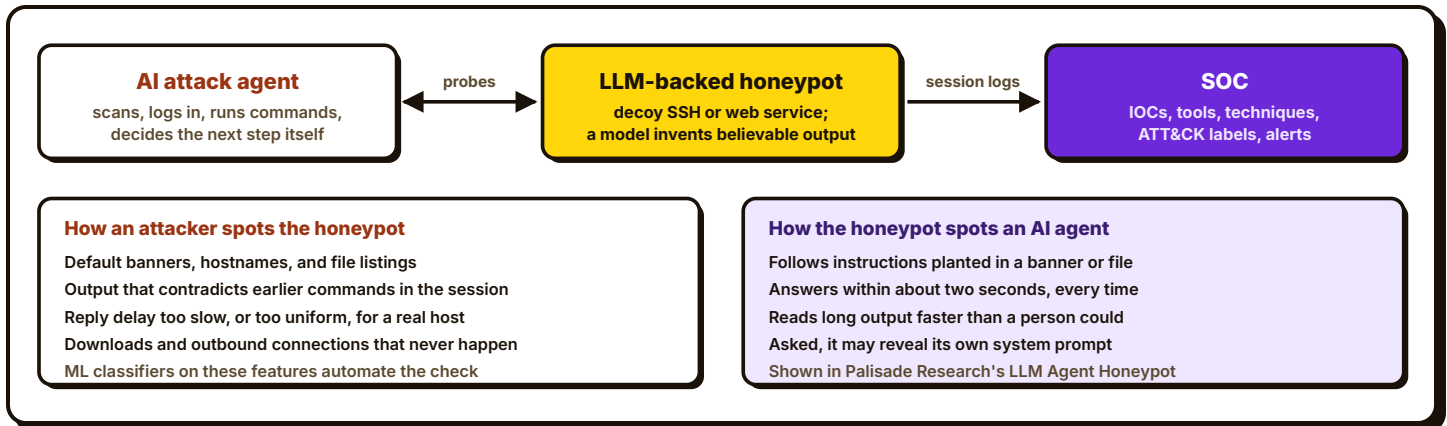


**FIGURE** Each side leaves tells. Defenders keep the decoy consistent and quick; attackers look for the seams.

### ■ Distributed denial of service (DDoS)

- Distributed denial of service (DDoS)**: many sources flood a target until real users cannot reach it
- How AI enhances it
  - Adaptive attacks: switch vector (volumetric, protocol, application layer) when one is mitigated
  - Bot traffic that imitates human browsing to slip past rate limits and CAPTCHAs
  - Timing picked from reconnaissance: the busiest sales hour
- AI endpoints are targets too: costly prompts cause a **denial of wallet** ( LLM10:2025 Unbounded Consumption )
- Defenses: ML traffic baselining in scrubbing and CDN services; bot management; per-identity rate limits and token quotas; spending caps; a tested DDoS runbook

| ATTACK VECTOR     | AI'S ROLE                     | EVIDENCE                    | DEFENSE                      |
|-------------------|-------------------------------|-----------------------------|------------------------------|
| Automated attacks | runs the kill chain           | fast, regular, parallel     | behavior detection           |
| Vector discovery  | maps, chains weaknesses       | scanning, API enumeration   | surface management, patching |
| Payloads, malware | variants, runtime model calls | AI API calls, odd processes | EDR, egress control          |
| Honeypot evasion  | fingerprints decoys           | sessions end at the tell    | consistent decoys            |
| DDoS              | adapts, mimics humans         | shifting traffic mix        | ML baselining, rate limits   |

## 3.2 Automated Attack Generation (continued)

### ON THE EXAM

- AI-assisted attacks are faster, larger versions of known techniques; prefer answers about behavior-based detection and patch speed over spotting "AI-written" code.
- Malware that calls a model at runtime leaves evidence (AI API traffic, embedded keys) that egress controls catch.
- Defenders use LLMs to make honeypots convincing and to spot AI agents; attackers fingerprint decoys by inconsistency and timing.

## 3.3 Automating Security Tasks with AI

### 3.0 AI-ASSISTED SECURITY

#### ■ Scripting tools, low-code, and no-code

- Three ways to build security automation, from most control to least effort
- **Scripting tools**: Python, PowerShell, Bash, and query languages (KQL, SPL)
  - An AI assistant drafts the script or query from a plain-language request
  - The analyst still owns it: read it, test it in a lab, run it with least privilege
  - Watch for invented functions, missing error handling, and hardcoded credentials
- **Low-code**: visual workflow builders with room for small code steps
  - SOAR playbook editors, cloud workflow services such as Azure Logic Apps (the engine behind Microsoft Sentinel playbooks), n8n
  - Many now include an "AI step" that calls a model in the middle of a flow
- **No-code**: drag-and-drop or natural-language builders for people who do not write code
  - "When a phishing report arrives, extract the URLs, check reputation, and reply to the user"
  - Fast to build; harder to review, version, and test

| APPROACH  | BUILT BY            | MAIN RISK                     |
|-----------|---------------------|-------------------------------|
| Scripting | engineers           | unreviewed AI-written code    |
| Low-code  | analysts, engineers | over-permissioned connectors  |
| No-code   | anyone              | shadow automation nobody owns |

#### ■ Document synthesis and summarization

- **Document synthesis and summarization**: condense or combine long text into what a reader needs
  - Incident timelines from chat, tickets, and logs
  - Shift handoff notes; threat intelligence reports turned into a list of IOCs and actions
  - Policy and audit evidence gathered into one summary for an assessor
- Risks
  - Omission: the one line that mattered is left out
  - Hallucination: details that were never in the source
  - Indirect prompt injection: instructions hidden in a document the model is asked to read
- Controls: link every statement to its source, keep the originals, and have a human approve summaries that leave the team

#### ■ Incident response ticket management

- **Incident response ticket management** is a good fit for AI
  - Turn raw alerts into tickets, merge duplicates, enrich with asset and user data, suggest severity and routing
- Never trust model output as data until it is checked
  - Constrain the format with a JSON schema
  - Validate the result, then check every value against the source (LLM05:2025 Improper Output Handling)

### 3.3 Automating Security Tasks with AI (continued)

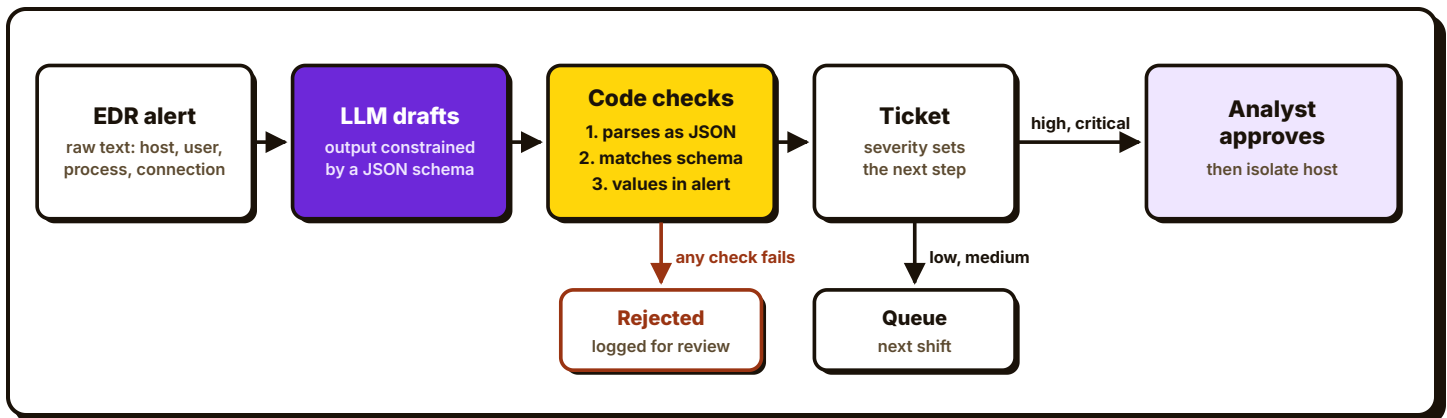


**FIGURE** The model drafts; code decides. Output that fails the schema or names something the alert never contained is rejected, and high-severity actions wait for an analyst.

#### TICKET.PY (EXCERPT)

```
def check(text):
    t = json.loads(text)           # is it JSON?
    validate(t, SCHEMA)           # right shape?
    for v in [t["host"], t["user"], *t["iocs"]]:
        if v not in ALERT:        # grounded?
            raise Ungrounded(v)
    return t
```

- Asked for a ticket in free text, qwen2.5:0.5b wrote Markdown and invented an event ID
- With the schema passed as Ollama's `format`, the same model returned valid JSON

#### QWEN2.5:0.5B

```
$ python3 ticket.py free
REJECTED JSONDecodeError
$ head -4 free.raw
Certainly! Below is an incident ticket for the EDR
alert you provided:

### Incident Ticket
**Event ID:** 12345
$ python3 ticket.py schema
action: isolate FIN-WS-114 -> PENDING analyst
approval
$ jq '.iocs' ticket.json
[
  "C:\\Users\\Public\\svc.exe",
  "powershell.exe",
  "10.20.4.17 -> 203.0.113.45:443"
]
$ python3 ticket.py check forged.json
REJECTED Ungrounded: not in alert: 198.51.100.9
```

- `forged.json` is the good ticket with one extra IP added, standing in for a hallucinated IOC

#### Change management and AI-assisted approvals

- **Change management:** every production change is requested, assessed, approved, scheduled, and recorded
  - Standard changes are pre-approved; normal changes go to review; emergency changes are fast-tracked and reviewed after
- AI helps at each step
  - Drafts the change request and the rollback plan from the commit or ticket
  - Scores risk from blast radius, past failures, and timing
  - Checks for conflicts with other scheduled changes
- **AI-assisted approvals:** the model recommends; a policy decides who must sign
  - Low risk, standard change: approved automatically, logged
  - High risk or production security control: a human approver, with the AI's reasoning attached
- The human approver stays accountable
  - Log the recommendation, the approver, and any override

#### Automated deployment/rollback

- **Automated deployment/rollback:** the pipeline releases a change and reverses it without waiting for a person
  - AI watches error rates, latency, and security alerts after a release and compares them with the baseline
  - A significant change triggers rollback to the last known good version, then a ticket
- Works for application code, firewall and WAF rules, detection content, and model versions
- Rehearse the rollback; an untested rollback is a second outage
  - The gates that decide when to deploy are in the next topic, AI in CI/CD

### 3.3 Automating Security Tasks with AI (continued)

#### AI agents

- An **AI agent** plans steps and calls tools to reach a goal with little human input
  - "Contain this host": query EDR, isolate the endpoint, disable the account, open the ticket, notify the owner
  - Tools are exposed through function calling or the **Model Context Protocol (MCP)**
- LLM06:2025 Excessive Agency is the risk: too much functionality, too many permissions, or too much autonomy
- Controls
  - Allowlist the tools; give each tool the narrowest scope (read-only where possible)
  - Human in the loop** for destructive or irreversible actions
  - Limits on steps, tokens, and spend per task
  - Log every tool call with its arguments for later audit

#### ON THE EXAM

- AI output that feeds another system must be validated (schema, allowlists, grounding checks) before it is used. Unvalidated output that triggers an action is improper output handling.
- AI-assisted approval means the model recommends and a human with authority approves high-risk changes. A choice that lets the model approve its own production changes is the distractor.
- An agent that can do more than its task needs is excessive agency; the fix is least-privilege tools and human approval for high-impact actions.

### 3.3 AI in CI/CD

#### 3.0 AI-ASSISTED SECURITY

#### CI/CD for AI systems

- Continuous integration and continuous deployment (CI/CD)**: every change is built, scanned, and tested automatically, then released by the pipeline
  - CI: merge small changes often; each push runs the checks
  - CD: a change that passes every gate ships without manual steps
- An AI system has more to change than code
  - Training data, model weights, prompts, retrieval sources, and dependencies
  - Each one gets a version and goes through the same pipeline
- AI also works inside the pipeline
  - Reviews pull requests, explains scanner findings, suggests fixes, writes tests
  - AI-written code goes through the same gates as human-written code

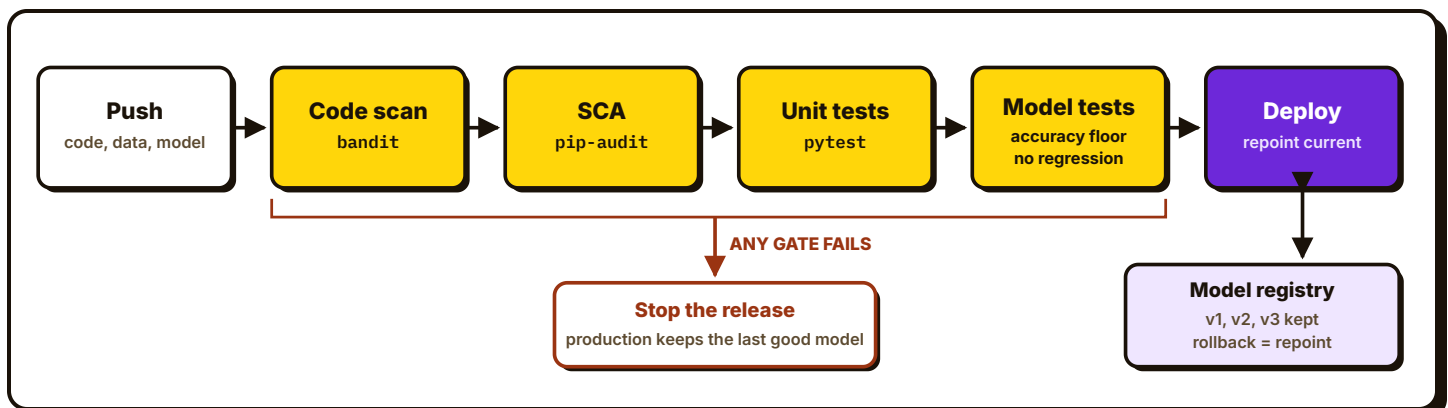


**FIGURE** Each gate can stop the release. A candidate model that fails a gate never replaces the one in production.

## Code scanning

- **Code scanning** (static application security testing, SAST) reads source code for insecure patterns without running it
  - Python: `bandit`, `semgrep`; many languages: CodeQL, SonarQube
  - Runs on every pull request; high-severity findings fail the build
- AI-specific patterns to catch
  - Loading models with `pickle` (code runs on load)
  - Model output passed to `eval`, a shell, or a SQL query
  - API keys for model providers written into the code

```
$ bandit -q -f custom bad/triage.py \
--msg-template "{line}: {test_id} {severity}"
1: B403 LOW
2: B404 LOW
4: B105 LOW
9: B301 MEDIUM
14: B602 HIGH
```

| ID          | FINDING                                                             |
|-------------|---------------------------------------------------------------------|
| B403 / B301 | <code>pickle</code> imported / used to load data                    |
| B404 / B602 | <code>subprocess</code> imported / run with <code>shell=True</code> |
| B105        | hardcoded password or token string                                  |

## Software composition analysis

- **Software composition analysis (SCA)** lists every third-party package and matches it against vulnerability databases
  - `pip-audit`, `npm audit`, OWASP Dependency-Check, Trivy, Gype
  - Produces or reads a **software bill of materials (SBOM)** in CycloneDX or SPDX format
- AI stacks pull in large dependency trees: ML frameworks, tokenizers, CUDA libraries, vector database clients
- Models are dependencies too (LLM03:2025 Supply Chain)
  - Pin model versions by hash; scan model files with a tool such as `modelscan`; prefer `safetensors` over pickle formats

### TERMINAL

```
$ cat bad/requirements.txt
torch==2.5.1
$ pip-audit -r bad/requirements.txt --no-deps \
--disable-pip -f json -o sca.json
$ jq -r -f cve.jq sca.json | sort -u
torch 2.5.1 PYSEC-2025-41 fix 2.6.0
```

- `cve.jq` keeps the advisory whose aliases include CVE-2025-32434
- CVE-2025-32434: `torch.load` with `weights_only=True` could still run code from a crafted model file; fixed in 2.6.0
  - The audit found many more PyTorch advisories for 2.5.1; the jq filter shows one

## Unit testing and regression testing

- **Unit testing**: small, fast tests of one function in isolation
  - Feature extraction, input validation, output parsers, prompt templates
- **Regression testing**: proves that what worked before still works
  - For code: the whole test suite reruns on every change
  - For models: the candidate must score at least as well as the model in production, within a small tolerance
- A model regression can also come from new data, a library upgrade, or a prompt change

### TESTS/TEST\_MODEL.PY (EXCERPT)

```
MIN_ACCURACY = 0.80 # absolute floor
MAX_DROP = 0.01 # vs the production model

def test_accuracy_threshold():
    assert CAND >= MIN_ACCURACY

def test_no_regression():
    prod = json.load(open("models/baseline.json"))
    assert CAND >= prod["accuracy"] - MAX_DROP
```

### 3.3 AI in CI/CD (continued)

#### Model testing

- **Model testing** checks the model artifact itself before release
  - Performance on a held-out set the model never trained on: accuracy, precision, recall
  - Per-class results; a login-risk model that misses attacks is worse than one with a few false alarms
  - Bias and fairness checks, robustness to noisy or adversarial input
  - For LLMs: evaluation sets, jailbreak and prompt-injection test suites (garak, promptfoo), and hallucination checks
- In the demo, v2 was retrained after a labeling bug flipped 35% of the training labels

#### SCIKIT-LEARN MODEL, SYNTHETIC LOGIN DATA

```
$ grep -E 'FAILED|passed' logs/model-tests.log
FAILED tests/test_model.py::test_no_regression -
assert 0.819 >= (0.842 - 0.01)
1 failed, 1 passed in 0.22s
```

- v2 clears the 0.80 floor, so a threshold alone would ship it
- The regression test catches the 2.3-point drop against v1

#### Automated deployment/rollback

- **Automated deployment**: the last pipeline stage releases the artifact once every gate passes
- **Automated rollback**: returns production to the last known good version
  - Before release: a failed gate leaves the current version in place
  - After release: health checks or canary metrics trigger the switch back
- Keep previous versions in an artifact store or model registry so rollback is a pointer change
- Release strategies that limit damage: canary (a small share of traffic first), blue/green (switch between two environments), shadow (new model scores live traffic, but its answers are not used)

#### TERMINAL

```
$ ./pipeline.sh v1
code-scan    PASS
sca          PASS
unit-tests   PASS
model-tests   PASS
deploy       models/current -> v1.json
$ ./pipeline.sh v2
code-scan    PASS
sca          PASS
unit-tests   PASS
model-tests  FAIL
deploy       SKIPPED, prod stays v1.json
```

#### ON THE EXAM

- Code scanning finds flaws in your own source; software composition analysis finds known vulnerabilities in the packages and models you depend on. A scenario naming a CVE in a library points to SCA.
- A model that still meets the minimum accuracy but scores lower than the current production model fails a regression test, not a threshold test.
- Rollback depends on keeping the previous version. "Restore service fastest after a bad model release" means redeploying the last known good artifact.

#### TRY IT

Run `pip-audit` against your own project's `requirements.txt`, then add `-f cyclonedx-json` to produce an SBOM from the same run.