NATIONAL SOFTWARE CONTEST 2026 — เอกสารประกอบการแข่งขัน

ShadowCat

AI-Powered Autonomous Penetration Testing Platform
(แบรนด์แดชบอร์ด: SecureThai)

เอกสารรวมทุกอย่างเกี่ยวกับโปรเจค: ภาพรวม / ประวัติการพัฒนา / เทคโนโลยี / สถาปัตยกรรมระบบ
คู่มือการนำเสนอ / แนวทางตอบคำถามกรรมการ / คำศัพท์เทคนิคที่ต้องรู้

จัดทำโดย Sarawut Thongthip | 13 กรกฎาคม 2026

สารบัญ

บทที่ 1 — ShadowCat คืออะไร 3

บทที่ 2 — พัฒนาโปรเจคนี้มาอย่างไรตั้งแต่ต้น 4

บทที่ 3 — เทคโนโลยีที่ใช้ทั้งหมด (Tech Stack) 6

บทที่ 4 — ระบบทั้งหมดมีกี่ระบบ อะไรบ้าง (ภาพรวมสถาปัตยกรรม) 8

บทที่ 5 — ระบบที่ 1: Backend (NSC2026 DAST Engine) 9

บทที่ 6 — ระบบที่ 2: Agent (CTF/HTB Agentic TUI) 13

บทที่ 7 — ระบบที่ 3: Frontend (SecureThai Dashboard) 15

บทที่ 8 — ระบบที่ 4: Benchmark (XBOW Validation) 16

บทที่ 9 — การทำงานแบบ End-to-End (Scan Lifecycle) 17

บทที่ 10 — กลไกความปลอดภัยของระบบเอง 18

บทที่ 11 — การทดสอบระบบ (Testing) 20

บทที่ 12 — คู่มือการนำเสนอ 21

บทที่ 13 — คู่มือการตอบคำถามกรรมการ + ตัวอย่างคำถาม 23

บทที่ 14 — คำศัพท์ทั้งหมดที่ควรรู้ 26

ภาคผนวก — คำสั่งที่ใช้บ่อย 31

หมายเหตุ: เลขหน้าเป็นค่าประมาณ ขึ้นอยู่กับการ render จริงของ PDF

บทที่ 1 — ShadowCat คืออะไร

1.1 ภาพรวมโปรเจค

ShadowCat คือแพลตฟอร์ม **AI-powered autonomous penetration testing** — ระบบทดสอบเจาะระบบความปลอดภัยเว็บแอปพลิเคชันแบบอัตโนมัติด้วย AI Agent ส่งเข้าแข่งขัน **National Software Contest 2026 (NSC2026)** รุ่นชื่อผลิตภัณฑ์/แดชบอร์ดที่ใช้คือ **SecureThai**

หัวใจของโปรเจคคือการสร้าง **enterprise-grade AI-driven DAST (Dynamic Application Security Testing)** ที่ทดสอบช่องโหว่ตาม OWASP Top 10 โดยอัตโนมัติ พร้อม "พิสูจน์" ทุกช่องโหว่ที่เจอด้วยกลไกตรวจสอบแบบ deterministic (กฎตายตัว ไม่ใช้ AI ตัดสิน) — จุดขายหลักคือ **"AI เก็บหลักฐาน มนุษย์/กฎเป็นคนตัดสิน"** เพื่อไม่ให้เกิดปัญหา AI hallucination (มั่วผลลัพธ์)

1.2 ปัญหาที่โปรเจคแก้

ปัญหาปัจจุบัน	รายละเอียด
ค่าใช้จ่ายสูง	จ้าง pentester มืออาชีพ 50,000–200,000 บาท ต่อการทดสอบ 1 ครั้ง
ใช้เวลานาน	1–2 สัปดาห์ต่อรอบการทดสอบ ทำให้ทดสอบซ้ำบ่อย ๆ ไม่คุ้ม
ขาดแคลนผู้เชี่ยวชาญ	บุคลากรด้าน pentest มีจำกัด โดยเฉพาะในไทย
เครื่องมือ AI ที่มีอยู่ไม่น่าเชื่อถือ	AI scanner ทั่วไปมักรายงานช่องโหว่ที่ไม่มีจริง (hallucination) ทำลายความเชื่อมั่นของผู้ใช้งาน

1.3 ทางแก้ของ ShadowCat

ShadowCat ใช้สถาปัตยกรรมแบบ **"actor / judge แยกกัน"**: AI Agent (LLM) ทำหน้าที่เป็นนักสืบ — ส่ง HTTP request เก็บหลักฐานเท่านั้น ห้ามตัดสินว่าเจอช่องโหว่จริงหรือไม่ ส่วน **Oracle** (โมดูล rule-based แยกต่างหาก ไม่ใช่ AI) เป็นผู้ตัดสิน (verdict) จากหลักฐานที่เก็บมา โดยเทียบกับ pattern ที่กำหนดไว้ตายตัว (เช่น SQL error string, เนื้อหาไฟล์ /etc/passwd, script tag ที่ reflect กลับมา) วิธีนี้ทำให้ผลลัพธ์ตรวจสอบย้อนกลับได้เสมอ (ทุก finding มี request_id อ้างอิงหลักฐานจริง)

> **เปรียบเทียบง่าย ๆ แบบ "ครัวร้านอาหาร":** API = พนักงานรับออเดอร์, Mode + SafetyPolicy = เมนู/กฎของร้าน (ทำได้แค่นี้ ห้ามทำเกินนี้), Orchestrator (ReAct loop) = หัวหน้าครัวที่คอยสั่งงาน, LLM = สมองของเชฟที่คิดว่าจะทำอะไรต่อ ส่วน Oracle คือ "หัวหน้าชิม" ที่ชิมแล้วให้คะแนนตามมาตรฐานตายตัว ไม่ใช่ตามใจเชฟ

1.4 สำหรับใคร (Use cases)

กลุ่มผู้ใช้	ประโยชน์
นักพัฒนา (Developer)	ทดสอบความปลอดภัยก่อน deploy ขึ้น production
ทีม Security ภายในองค์กร	สแกนได้บ่อยขึ้นโดยไม่ต้องจ้างนอกทุกครั้ง

ทีม Compliance	ออกรายงานอ้างอิง PDPA / ISO 27001 ได้อัตโนมัติ (โมดูลนี้ยังอยู่ระหว่างพัฒนา)
Bug bounty hunter	ค้นหาช่องโหว่ได้เร็วขึ้นในขอบเขตที่ได้รับอนุญาต

1.5 ผลการทดสอบมาตรฐาน (Benchmark)

ระบบผ่านการทดสอบด้วย **XBOW validation benchmark suite** จำนวน 104 โจทย์ (ครอบคลุมช่องโหว่ 15+ ประเภท เช่น XSS, IDOR, SQLi, SSTI, command injection, LFI, JWT, SSRF ฯลฯ)

ระดับความยาก	อัตราสำเร็จ
Level 1 (ง่าย)	91.1% (50/55)
Level 2 (ปานกลาง)	74.5% (35/47)
Level 3 (ยาก)	62.5% (5/8)
รวม	**86.5% (90/104)**

ค่าใช้จ่ายเฉลี่ย **~$1.11 ต่อ scan** (median $0.42) ใช้เวลาเฉลี่ย **~6.1 นาที** ต่อ scan — เทียบกับ pentest แบบ manual ที่ $5,000+ และ 1–2 สัปดาห์ ถูกกว่าประมาณ **100 เท่า**

1.6 ข้อควรระวังในการนำเสนอ (สิ่งที่ยังไม่เสร็จ/เป็น skeleton)

> เพื่อความน่าเชื่อถือเวลาตอบกรรมการ ควรรู้ตรงไปตรงมาว่าส่วนใดยัง**ไม่ได้ wire เข้าใช้งานจริง**:
>
> - **WAF evasion** (`backend/waf/evasion.py`) — เขียนโค้ด mutation ไว้แล้ว แต่ยังไม่ได้เรียกใช้จาก orchestrator
> - **CTF mode ใน backend/** (`backend/modes/ctf.py`) — เป็น skeleton (โยน NotImplementedError) การแก้โจทย์ CTF จริงอยู่ใน package `agent/` แยกต่างหาก
> - **PDPA compliance mapping** (`backend/compliance/pdpa_mapping.py`) — เป็น skeleton เช่นกัน
> - **nuclei_tool.py** — วางโครงไว้ 1 บรรทัด ยังไม่ implement
>
> ตอบกรรมการแบบยอมรับตรง ๆ ว่า "อยู่ในแผนพัฒนาต่อ" จะดูน่าเชื่อถือกว่าการกลบเกลื่อน

บทที่ 2 — พัฒนาโปรเจคนี้มาอย่างไรตั้งแต่ต้น

2.1 หมายเหตุสำคัญเรื่อง Git History

ตัว repository ถูก "reset" ใหม่แบบสะอาด (commit แรกชื่อ `"Initial commit - Clean start for NSC competition"`) เพื่อการส่งแข่งขันโดยเฉพาะ ดังนั้น git log จึงมีแค่ 5 commit หลัก ไม่ได้สะท้อนพัฒนาการทั้งหมด — เรื่องราวการพัฒนาแบบละเอียดจริง ๆ อยู่ใน `docs/PROGRESS.md` ซึ่งเป็น dev log บันทึกเป็นรอบ ๆ (session) ตั้งแต่ 2026-06-06 ถึงปัจจุบัน

2.2 ไทม์ไลน์ระดับ Git Commit

Commit	วันที่	สิ่งที่เกิดขึ้น
`6278a4e`	2026-06-02	Initial commit — reset repo ใหม่สำหรับการแข่งขัน ตอนนั้นยังมี package `api/` (ต้นแบบ pipeline แบบ DAG หลาย-LLM รุ่นเก่า) และ agent ชื่อ `pentestgpt/`
`58403b5`	2026-06-12	Refactor ใหญ่: ลบ `api/` (DAG pipeline) ทิ้งทั้งหมด, เปลี่ยนชื่อ `pentestgpt/` → `agent/` — จุดกำเนิดโครงสร้าง 2 package หลัก `backend/` + `agent/` ที่ใช้อยู่ปัจจุบัน
`f54e528`	2026-06-12	เพิ่ม `victim_markers` สำหรับยืนยัน IDOR, ปิดช่องโหว่ SSRF จาก auto-follow-redirect, เพิ่ม verdict ระดับ "likely" ใน GeneralOracle, ปรับปรุง WAF evasion case-handling, เพิ่มเทสต์จำนวนมาก
`ec3d3df`	2026-07-02	ลบไฟล์ demo (gif/mp4/cast) ที่ไม่จำเป็นออกจาก repo เพื่อให้ขนาดกระชับสำหรับส่งแข่ง
`2e00a2e`	2026-07-06	แก้ submodule + gitignore, เพิ่มระบบ "Local-First Agent OS" (daemon/cli/authz/workspace) เข้า git อย่างเป็นทางการ, เพิ่ม `docs/ROADMAP.md`

งานล่าสุด ณ วันที่ทำเอกสารนี้ (2026-07-13) เช่น แดชบอร์ด HeroUI v3 ใหม่ทั้งหมดใน `frontend/` และเอกสารภาษาไทยชุดนี้ ยังอยู่ในสถานะ "แก้ไขแล้วแต่ยังไม่ commit" (working tree) — ยังไม่ถูกบันทึกเป็น commit ใหม่

2.3 พัฒนาการตามช่วง (Phase) จาก docs/PROGRESS.md

นี่คือลำดับการพัฒนาแบบละเอียดกว่า git log มาก เพราะบันทึกทุก session การพัฒนาจริง:

ช่วงที่ 1 — วางรากฐาน DAST Engine (session 2026-06-06)

Phase	สิ่งที่สร้าง
Phase 1	Authenticated Crawling + CSRF token extraction/replay
Phase 2	WAF Detection & Evasion (fingerprint 9 ยี่ห้อ + mutation techniques)
Phase 3	JavaScript API Discovery — ดึง endpoint จากไฟล์ JS bundle
Phase 4	Subdomain enumeration ผูกเข้ากับ scope ของการสแกน

Phase 5	Context Pruning — ตัดประวัติบทสนทนาเพื่อลดต้นทุน LLM
Phase 6	Concurrent Spider Workers — crawl แบบขนานหลาย worker
Phase 7	Directory Brute-forcing (wordlist ~180 path)
Phase 8	HTML/PDF Report generation

ช่วงที่ 2 — Local-First Agent OS / "dashboard moment" (session 2026-06-13)

ช่วงนี้คือการเปลี่ยนจากระบบที่ต้องรันด้วยคำสั่งเทอร์มินัลอย่างเดียว ให้กลายเป็นแพลตฟอร์มที่มี daemon พื้นหลัง + แดชบอร์ดเว็บ:

- **Phase 1 "dashboard moment"** — เริ่มสร้างแดชบอร์ดเว็บตัวแรกสำหรับดู scan แบบ real-time
- **Phase 2 self-served dashboard** — daemon เสิร์ฟแดชบอร์ดเองได้ ไม่ต้องพึ่งเซิร์ฟเวอร์แยก
- **Phase 3 OpenClaw-style UI + workspace layer** — เพิ่มระบบ "workspace" (ไฟล์ config เป็น markdown แก้ไขได้ เช่น SOUL.md)
- **Phase 4 readiness** — เชื่อม workspace เข้าระบบจริง + เพิ่ม authorization gate (ห้าม scan เป้าหมายนอกขอบเขตที่อนุญาต)

งานเสริมในช่วงเดียวกัน: แก้บั๊ก "agent รายงานผลช้าเกินไป" ให้ agent รายงาน finding ทันทีที่เจอ, เพิ่ม headless browser crawler (Playwright) รองรับเว็บที่เป็น SPA ที่ต้อง render JS ก่อนถึงจะเห็นเนื้อหา (เรียกใน docs ว่า "L3→L4 upgrade"), เพิ่ม Memory system ที่มองเห็นได้จากแดชบอร์ด, ปรับปรุง Live Feed UX ให้เห็น token/cost แบบ real-time, และไล่แก้บั๊ก SSE ที่ทำให้หน้าจอค้างที่สถานะ "Queued"

ช่วงที่ 3 — เก็บงาน (session 2026-06-12 ถึง 2026-07-12)

ปรับปรุงต่อเนื่อง: แก้ปัญหา PSU Blue gateway บล็อก User-Agent ของ OpenAI SDK, แก้ปัญหาการรั่วไหลของ environment variable เข้าไปใน test, แก้บั๊ก HeroUI v3 บน modal และ table component, เพิ่มแท็บ Memory/Settings/Live-Feed ในแดชบอร์ด, เพิ่มระบบตั้งค่าที่แก้ไขได้ผ่านหน้าเว็บ (`user_config.py`)

2.4 ขั้นตอนถ้าจะเริ่มโปรเจคใหม่ตั้งแต่ศูนย์ (สรุปเป็นแนวคิด)

1. ออกแบบสถาปัตยกรรม "actor แยกจาก judge" ก่อนเขียนโค้ดบรรทัดแรก (ดู `docs/ENTERPRISE_ARCH.md` ซึ่งเขียนเป็น design brief ล่วงหน้า)
2. สร้าง ReAct loop กลาง (`Orchestrator`) ที่ mode-agnostic — ใช้ได้กับหลายโหมดการสแกน
3. สร้าง **SafetyPolicy** และ **EvidenceStore** ก่อนต่อ tool ใด ๆ เข้ากับ LLM เพื่อไม่ให้เกิดช่องโหว่จากตัวเอง (SSRF, out-of-scope request)
4. สร้าง Oracle (rule-based verifier) แยกจาก Agent อย่างเด็ดขาด
5. ต่อ Crawler / WAF detector / Subdomain enum เป็น pre-scan phase ก่อนปล่อย Agent ทำงาน เพื่อให้ agent มี "แผนที่" ก่อนเริ่ม
6. เพิ่ม daemon + CLI + dashboard ให้ใช้งานง่ายแบบ local-first (ไม่ต้องพึ่ง cloud)
7. เพิ่ม memory ข้าม scan, cost control, testing suite ให้ครบตามหลัง

บทที่ 3 — เทคโนโลยีที่ใช้ทั้งหมด (Tech Stack)

3.1 ภาษาและ Backend หลัก

เทคโนโลยี	เวอร์ชัน / รายละเอียด
Python	>=3.12,<4.0
ตัวจัดการ dependency	`uv` (astral-sh) + `uv.lock`
Build backend	`hatchling`
Web framework	FastAPI >=0.110.0 + Uvicorn (ASGI server)
Data validation	Pydantic >=2.5.0 / pydantic-settings
HTTP client	httpx >=0.27
HTML parsing	BeautifulSoup4 + lxml
Database ORM	SQLAlchemy[asyncio] >=2.0.30 + asyncpg (PostgreSQL, เปิดใช้แบบ opt-in)
TUI framework	Textual >=0.47.0 (ใช้ใน `agent/` ทำ terminal UI)
CLI แสดงผลสวยงาม	Rich >=13.7.0
Headless browser (optional)	Playwright >=1.60.0 (extra: `headless`)
Telemetry	Langfuse >=2.0.0
PDF rendering (optional)	WeasyPrint (ถ้าไม่มี ตัวระบบ fallback ไปสร้าง HTML แทน)

3.2 LLM ที่รองรับ (Multi-provider)

Provider	ลักษณะ Key	หมายเหตุ
OpenRouter	`sk-or-...`	ค่าเริ่มต้นของระบบ เข้าถึงหลายโมเดลผ่าน gateway เดียว
PSU Blue (ม.สงขลานครินทร์)	`sk-user-...`	gateway แบบ vLLM, ตรวจจับ prefix key อัตโนมัติ, บังคับ streaming เสมอ, LLMClient ต้อง override User-Agent เพราะ Cloudflare WAF ของ ai.psu.blue บล็อก UA ของ OpenAI SDK
Anthropic	API key ตรง	เรียก Claude โดยตรง
Claude subscription (OAuth)	ผ่าน Claude Agent SDK	ต้องมี npm CLI `@anthropic-ai/claude-code`
Local LLM	ไม่ต้องมี key	Ollama / LM Studio / text-generation-webui ผ่าน OpenAI-compatible endpoint

3.3 Frontend (SecureThai Dashboard)

เทคโนโลยี	เวอร์ชัน
Next.js	^16.2.10 (App Router, static export mode)
React / react-dom	^19.2.7
HeroUI	@heroui/react ^3.2.2, @heroui/styles ^3.2.2
Tailwind CSS	^4.3.2 (+ @tailwindcss/postcss)
TypeScript	^5.9.3

แดชบอร์ดถูก build เป็น static export ไปที่ `frontend/out` แล้วให้ backend (FastAPI) เสิร์ฟไฟล์เหล่านี้เองที่ path `/` — จบในกระบวนการเดียว พอร์ตเดียว (8000) ไม่ต้องมี Node server แยกตอนใช้งานจริง

3.4 Docker / Infra

รายการ	รายละเอียด
Base image	Ubuntu 24.04
เครื่องมือติดตั้งในอิมเมจ	nmap, netcat-openbsd, curl, wget, git, openvpn, jq, ripgrep, tmux, dnsutils, whois
Node.js	เวอร์ชัน 20 + npm CLI `@anthropic-ai/claude-code`
ผู้ใช้ในคอนเทนเนอร์	non-root user ชื่อ `shadowcat` พร้อม sudo แบบไม่ต้องใส่รหัส
docker-compose	service `shadowcat`, cap_add NET_ADMIN + mount /dev/net/tun (สำหรับต่อ VPN ของ HTB/THM), จำกัดทรัพยากร 4 CPU / 8GB

3.5 คุณภาพโค้ด / CI / Testing

เครื่องมือ	ใช้ทำอะไร
ruff	Linter + formatter (target py312, line-length 100)
mypy	Type checking แบบ strict
pytest + pytest-asyncio + pytest-cov	รันเทสต์ + coverage
GitHub Actions CI	4 jobs: lint, typecheck, test, test-docker, build

บทที่ 4 — ระบบทั้งหมดมีกี่ระบบ อะไรบ้าง

โปรเจคแบ่งออกเป็น **4 ระบบหลัก** ที่ทำงานอย่างชัดเจนแยกจากกัน บวกกับ 1 ระบบเสริม (benchmark ใช้ทดสอบมาตรฐาน):

#	ระบบ	โฟลเดอร์	หน้าที่หลัก	ผู้ใช้งานหลัก
1	**Backend — NSC2026 DAST Engine**	`backend/`	เครื่องยนต์หลักของการแข่งขัน: สแกนเว็บแบบอัตโนมัติ + verify หลักฐาน + ออกรายงาน	ผู้ประเมิน/กรรมการแข่งขัน (ระบบหลักที่ใช้ตัดสิน)
2	**Agent — CTF/HTB Agentic TUI**	`agent/`	Agent แบบ terminal UI สำหรับแก้โจทย์ CTF / HackTheBox โดยเฉพาะ (ยิง shell จริงได้)	ผู้ใช้ที่ต้องการ auto-solve โจทย์แข่งขันแฮก
3	**Frontend — SecureThai Dashboard**	`frontend/`	เว็บแดชบอร์ดแสดงผลการสแกนแบบ real-time, จัดการ scan, ดู memory/settings	ผู้ใช้ทั่วไปที่ไม่อยากใช้ command line
4	**Benchmark — XBOW Validation**	`benchmark/`	รันชุดทดสอบมาตรฐาน 104 โจทย์ เพื่อวัดความแม่นยำของ agent	ทีมพัฒนา ใช้วัดผลก่อนส่งแข่ง

> **ข้อแตกต่างสำคัญระหว่างระบบที่ 1 (Backend) กับระบบที่ 2 (Agent):**
> Backend ถูกออกแบบให้ **ปลอดภัย/มีขอบเขต** — ยิงได้แค่ HTTP, ห้ามใช้ shell, ทุกช่องโหว่ต้องผ่าน Oracle ยืนยัน เหมาะกับงาน pentest จริงที่ต้องมีหลักฐานอ้างอิงได้
> Agent ถูกออกแบบให้ **ยืดหยุ่น/รุกได้เต็มที่** — เปิด terminal จริง รัน exploit script CVE ได้ เหมาะกับการแข่งแก้โจทย์ CTF ที่เป้าหมายคือ "เจอ flag" ไม่ใช่การพิสูจน์ความปลอดภัยแบบมีหลักฐาน
> ทั้งสองระบบเป็นคนละ codebase แทบจะอิสระจากกัน ไม่ได้ใช้ Orchestrator ตัวเดียวกัน

4.1 แผนภาพโครงสร้างโฟลเดอร์ระดับบนสุด

```
backend/     ระบบหลัก NSC2026 — enterprise AI DAST (FastAPI + SSE)
agent/       Agentic TUI (CTF/HTB) — คำสั่ง shadowcat-agent
frontend/    SecureThai dashboard (Next.js 16 + HeroUI) — build แล้วให้ backend เสิร์ฟ
benchmark/   ชุดทดสอบมาตรฐาน XBOW (104 โจทย์)
docs/        เอกสารสถาปัตยกรรม, progress log, สไลด์นำเสนอ
tests/       ชุดทดสอบอัตโนมัติของทั้งโปรเจค
scripts/     สคริปต์ container + เครื่องมือช่วยพัฒนา
```

บทที่ 5 — ระบบที่ 1: Backend (NSC2026 DAST Engine)

นี่คือระบบที่ใช้แข่งขันจริง เขียนด้วย FastAPI แบ่งออกเป็นโมดูลย่อยชัดเจน 10 กลุ่ม

5.1 โครงสร้างโมดูลย่อยของ backend/

โมดูล	หน้าที่
`api/routes_scan.py`	FastAPI app หลัก รวม endpoint ทั้งหมด (13 เส้นทาง)
`core/`	**orchestrator.py** = ReAct loop กลาง, **llm_client.py** = ตัวเรียก LLM แบบ multi-provider, **events.py** = SSE event, **coverage.py** = กันทดสอบซ้ำ, **scan_memory.py** = ความจำข้าม scan
`crawler/`	spider.py (BFS แบบขนาน), csrf.py, js_analyzer.py, subdomains.py, headless.py (Playwright)
`modes/`	base.py (SafetyPolicy), enterprise.py, general.py, ctf.py (skeleton), registry.py
`tools/`	http_tool.py, report_tool.py, general_report_tool.py, dirbrute_tool.py, nuclei_tool.py (skeleton)
`verification/`	evidence_store.py, idor_oracle.py, general_oracle.py
`waf/`	detector.py (ต่อใช้งานแล้ว), evasion.py (ยังไม่ได้ต่อใช้งาน)
`reporting/generator.py`	สร้างรายงาน HTML/PDF
`db/`	schema.sql (7 ตาราง), models.py, engine.py, repository.py — ใช้ PostgreSQL แบบ opt-in ผ่าน DATABASE_URL
`compliance/pdpa_mapping.py`	โครงสร้างสำหรับ mapping ตาม พ.ร.บ.คุ้มครองข้อมูลส่วนบุคคล (ยังเป็น skeleton)
`schemas/api.py`	โมเดล Pydantic ทั้งหมดที่ใช้สื่อสารผ่าน API
`daemon.py / cli.py / authz.py / workspace.py / user_config.py`	ชั้น "Local-First Agent OS": จัดการ daemon เบื้องหลัง, คำสั่ง `shadowcat`, ประตูตรวจสอบสิทธิ์, ไฟล์ config แบบแก้ไขได้, การตั้งค่าผู้ใช้

5.2 API Endpoints ทั้งหมด

Method & Path	ทำอะไร
`POST /api/scans`	สร้าง+คิว+เริ่ม scan ใหม่ ผ่านการตรวจสิทธิ์ (authz) ก่อน คืนค่า 202 พร้อม scan_id
`GET /api/scans`	ดูรายการ scan ที่กำลังทำงาน (สำหรับ sidebar)

`GET /api/scans/{id}/stream`	สตรีมผลแบบ real-time ด้วย Server-Sent Events (SSE) รองรับ resume ผ่าน header `Last-Event-ID`
`GET /api/scans/{id}`	เช็คสถานะ/ผลลัพธ์แบบ poll ได้ (fallback ไปอ่านจาก DB ถ้าหลุดจากความจำ)
`GET /api/scans/{id}/evidence/{request_id}`	ดึงคู่ request/response ดิบ — ต้องมี bearer token ยืนยันตัวตน
`GET /api/scans/{id}/report?fmt=html|pdf`	สร้างรายงาน (fallback PDF→HTML ถ้าไม่มี renderer)
`POST /api/scans/{id}/cancel`	ยกเลิก scan แบบ cooperative (เช็คสถานะ ณ จุดสิ้นสุด turn ถัดไป)
`GET /health`	เช็คว่า daemon ยังทำงานอยู่ (ไม่ต้อง auth)
`GET /api/workspace, /api/workspace/{name}`	ดูรายชื่อ/อ่านไฟล์ config-as-markdown ที่แก้ไขได้
`GET /api/memory`	ดูความจำข้าม scan แยกตาม host
`GET /api/settings, PUT /api/settings`	อ่าน/แก้ไขค่า config (API key แบบปิดบัง, model ที่ใช้)
`GET /, /{path}`	เสิร์ฟหน้าเว็บแดชบอร์ด (frontend/out ถ้ามี ไม่งั้น fallback ไป webui/index.html)

5.3 ReAct Loop / Orchestrator — หัวใจของระบบ

ReAct = Reasoning + Acting คือรูปแบบที่ให้ AI คิด → ทำ (เรียก tool) → ดูผล → คิดต่อ วนซ้ำจนกว่าจะจบงาน

```
Turn 1: LLM คิด "ต้องหา endpoint ก่อน" → เรียก dir_bruteforce()
        → พบ /admin, /api/users
Turn 2: LLM คิด "/api/users น่าสนใจ ลองทดสอบ SQLi"
        → เรียก http_request(GET /api/users?id=')
        → ได้ HTTP 500 + ข้อความ error SQL
Turn 3: LLM คิด "เจอ SQL error → รายงานเป็นหลักฐาน"
        → เรียก report_finding(request_id=...)   ** ไม่ตัดสินเองว่าใช่ช่องโหว่ **
-------------------------------------------------------------
หลังจบ loop: Oracle ดึงหลักฐานจาก request_id มาตรวจตาม pattern
             ตัดสิน verdict: confirmed / likely / inconclusive / refuted
```

5.3.1 ขั้นตอนแบบละเอียด (จาก create scan ถึง report)

1. **POST /api/scans** — ตรวจ scope ที่อนุญาต (authz), สร้าง SafetyPolicy, ทำ auto-login ถ้ามีการตั้งค่า identity (username/password), เลือก Mode (enterprise/general/ctf), คืนค่า scan_id ทันทีแล้วรันงานจริงแบบ background task
2. **Pre-scan Phase (ก่อน AI เริ่มทำงาน)**
   - Phase 0a — Subdomain enumeration (ถ้าเปิดใช้)
   - Phase 0b — WAF detection (เปิดใช้เป็นค่าเริ่มต้น) — ยิง probe รู้ทันทีว่าใช้ Cloudflare/Sucuri/ฯลฯ

- Phase 0c — Spider crawl แบบ BFS ขนาน — หา endpoint/form/JS API ก่อน แล้วสรุปเป็น "attack surface" ป้อนให้ agent

3. **Orchestrator.run()** — ลูป ReAct จริง: จำกัด turn สูงสุด (ค่าเริ่มต้น 40 turn), ทุก ๆ 6 turn จะแจ้งเตือนให้ agent อย่าทดสอบซ้ำจุดเดิม (Coverage Ledger), เรียก tool ได้พร้อมกันหลายตัวต่อ 1 turn, ทุกการเรียก tool ต้องผ่านการตรวจ SafetyPolicy ก่อนเสมอ
4. **Verification Phase** — เมื่อ loop จบ ระบบเปลี่ยนสถานะเป็น "verifying" แล้วส่งทุก finding ที่ agent รายงานไปให้ Oracle ตัดสินจากหลักฐานจริงใน Evidence Store
5. **บันทึกผล** — เก็บลง PostgreSQL (ถ้าตั้งค่าไว้), อัปเดตความจำข้าม-scan ต่อ host
6. **Report** — สร้างรายงาน HTML/PDF ตามคำขอ

5.4 Modes และ SafetyPolicy

Mode	HTTP Methods ที่อนุญาต	ทำ state-changing ได้ไหม	Tools ที่ใช้ได้	สถานะ
enterprise	GET, HEAD	ไม่ได้	http_request, report_finding	ใช้งานได้จริง — เน้น IDOR
general	GET, HEAD, POST, OPTIONS	ได้	dir_bruteforce, http_request, report_finding	ใช้งานได้จริง — ครอบคลุม OWASP Top 10
ctf (ใน backend/)	—	ไม่ได้	—	skeleton ยังไม่ implement (ของจริงอยู่ที่ package agent/)

SafetyPolicy คือ dataclass ที่กำหนดขอบเขต (scope) และวิธี (method) ที่ agent ทำได้ ถูกตรวจสอบทุกครั้งก่อน tool จะทำงานจริง (ไม่ใช่แค่บอก agent เฉย ๆ) นอกจากนี้ยังบล็อก cloud metadata host (เช่น `169.254.169.254`) ตายตัว เพื่อกัน SSRF ไปยัง internal cloud metadata แม้ scope จะครอบคลุมก็ตาม

5.5 เครื่องมือ (Tools) ที่ Agent เรียกใช้ได้

Tool	หน้าที่
`http_request`	ยิง HTTP request ตาม scope ที่อนุญาต — ตัดข้อมูล credential ที่ agent พยายามใส่เองออกทั้งหมด แล้ว inject ของจริงจากฝั่งเซิร์ฟเวอร์แทน
`dir_bruteforce`	ค้นหา path/ไดเรกทอรีที่ซ่อนอยู่ (เฉพาะ general mode) ด้วย wordlist ~180 path
`report_finding` / `report_general_finding`	ให้ agent "รายงาน" ช่องโหว่ที่สงสัย — ต้องอ้างอิง request_id ที่มีอยู่จริงใน Evidence Store เท่านั้น ห้ามใส่ field สถานะ/severity เอง (schema บังคับ)

5.6 Evidence Store & Oracle — กลไกป้องกัน AI มั่ว

Evidence Store เก็บทุกคู่ request/response แบบ append-only พร้อม ID อ้างอิง (r1, r2, ...) — LLM เห็นแค่ตัวอย่างเนื้อหาสั้น ๆ (~900 ตัวอักษร) ไม่เห็น response เต็ม เพื่อลดต้นทุนและลดโอกาสให้ agent "จำผิด/เดาเอง"

Oracle คือโมดูล rule-based (ไม่ใช้ AI) ที่ดึงหลักฐานจริงจาก Evidence Store มาตรวจตาม pattern ที่กำหนดไว้ตายตัว มี 2 ตัว:

- **IdorOracle** — ใช้กับ enterprise mode ตรวจสอบ IDOR โดยเทียบ response ของ "baseline" (เจ้าของข้อมูลเข้าถึงข้อมูลตัวเอง) กับ "cross-access" (ผู้ใช้ A พยายามเข้าข้อมูลผู้ใช้ B)
- **GeneralOracle** — ใช้กับ general mode ตรวจ pattern ของ SQLi (18 รูปแบบ error message ต่างฐานข้อมูล), Path Traversal (7 marker เช่น /etc/passwd), XSS (12 indicator), sensitive data leak, blind SQLi (จับเวลา response), open redirect

ระดับผลตัดสิน (verdict): confirmed เจอ pattern ชัดเจน · likely (เฉพาะ general) น่าจะใช่แต่ยังไม่ฟันธง · inconclusive ไม่พอสรุป · refuted พิสูจน์แล้วว่าไม่ใช่

5.6.1 กรณีศึกษา: victim_markers สำหรับยืนยัน IDOR

ฟีเจอร์นี้ถูกเพิ่มเข้ามาเฉพาะเพื่อแก้ปัญหา "รู้ว่าเข้าถึงข้อมูลคนอื่นได้ แต่พิสูจน์ไม่ได้ว่าเป็นข้อมูลของเขาจริง" — ผู้ควบคุมการสแกนใส่ `victim_markers` (เช่น อีเมลของผู้ใช้ B) เข้าไปในคำขอสแกน ระบบจะตรวจว่า marker เหล่านี้:

1. ปรากฏใน response ตอนที่ผู้ใช้ A ขอเข้าถึงข้อมูลของผู้ใช้ B (cross-access) **และ**
2. **ไม่ปรากฏ**ใน response ตอนที่ผู้ใช้ A เข้าถึงข้อมูลของตัวเอง (baseline)

ถ้าตรงทั้งสองเงื่อนไข → confirmed ทันที เพราะพิสูจน์ได้ทางคณิตศาสตร์ว่าเป็นข้อมูลของ B จริง ไม่ใช่การเดา — นี่คือหัวใจของคำว่า "evidence-grounded" ที่ทำให้ผลลัพธ์น่าเชื่อถือกว่า AI ทั่วไป

5.7 WAF Detection

ยิง probe payload ที่รู้อยู่แล้วว่า "อันตราย" (เช่น `' OR '1'='1` ผสม script tag) แล้วเทียบ header/body ของ response กับลายเซ็นของ WAF 9 ยี่ห้อ (Cloudflare, Sucuri, Imperva, F5 BIG-IP, AWS WAF, ModSecurity, Barracuda, FortiWeb, generic) ผลลัพธ์คือ `WafProfile` พร้อมค่าความมั่นใจ (confidence) — ส่วนนี้ต่อใช้งานจริงแล้ว ส่วน evasion (การหลบเลี่ยง WAF) ยังเขียนไว้แต่ไม่ได้ต่อเข้า orchestrator

5.8 Reporting, DB, Compliance

Reporting (`backend/reporting/generator.py`) สร้างรายงานแบบ self-contained HTML พร้อม severity badge และตัวอย่างหลักฐาน — ถ้าขอ PDF ระบบจะลองใช้ WeasyPrint ก่อน ถ้าไม่มีจะลอง headless Chromium ผ่าน Playwright ถ้าไม่มีอีกจะ fallback เป็น HTML

Database เป็น PostgreSQL (async SQLAlchemy) ใช้แบบ opt-in ผ่าน environment variable `DATABASE_URL` มี 7 ตาราง — ถ้าไม่ตั้งค่าไว้ ระบบยังทำงานได้ปกติแต่ไม่บันทึกถาวร

Compliance (`backend/compliance/pdpa_mapping.py`) เป็นแผนสำหรับ mapping ผลสแกนเข้ากับข้อกำหนดใน พ.ร.บ.คุ้มครองข้อมูลส่วนบุคคล (PDPA) — **ปัจจุบันยังเป็น skeleton** ยังไม่ implement จริง

บทที่ 6 — ระบบที่ 2: Agent (CTF/HTB Agentic TUI)

เป็น package แยกต่างหากจาก backend เดิมชื่อ `pentestgpt/` ก่อนถูกเปลี่ยนชื่อเป็น `agent/` ออกแบบมาสำหรับแก้โจทย์ style CTF/HackTheBox/THM ผ่าน terminal UI แบบโต้ตอบได้ (สั่ง `shadowcat-agent --target <IP/URL>`)

6.1 องค์ประกอบหลัก

โมดูล	หน้าที่
`core/controller.py`	**AgentController** — วงจรชีวิต 5 สถานะ: IDLE → RUNNING → PAUSED → COMPLETED → ERROR ควบคุมผ่าน pause()/resume()/stop()/inject()
`core/backend.py`	ตัวเชื่อม LLM backend 3 แบบ: OpenRouterBackend (ค่าเริ่มต้น), AnthropicBackend, ClaudeSDKBackend (OAuth)
`core/events.py`	**EventBus** — ตัวกลาง pub/sub แบบ singleton เชื่อม TUI กับ agent แบบไม่ผูกกันตรง ๆ
`core/session.py`	**SessionStore** — บันทึก session ลงไฟล์ JSON ที่ `~/.shadowcat/sessions/` เพื่อ resume งานเดิมได้
`core/planner.py`	เรียก LLM ราคาถูก 1 ครั้งก่อนเริ่ม loop หลัก เพื่อวางแผน exploit chain ล่วงหน้าเป็น JSON
`core/phantom.py`	โมดูล web-fingerprinting ขนาดใหญ่ที่สุดในโปรเจค (~2,300 บรรทัด) ใช้ตรวจสอบเทคโนโลยีของเป้าหมาย
`interface/tui.py`	แอป Textual ชื่อ `ShadowCatApp` — แสดง activity feed แบบ real-time, กด F1 ดู help, Ctrl+P pause, Ctrl+Q quit
`prompts/pentesting.py`	**CTF_SYSTEM_PROMPT** — system prompt ขนาดใหญ่ ครอบคลุมวิธีคิดแบบ CTF, รูปแบบ flag, กลยุทธ์สำรอง, และการป้องกัน prompt injection จากผลลัพธ์ของ tool
`tools/`	BaseTool/TerminalTool (รัน shell จริง), AsyncToolExecutor, ToolRegistry
`parsing/html_distiller.py`	ตัด script/style ออกจาก HTML เพื่อลด token ลง 30-50 เท่า

6.2 การตรวจจับ Flag (Flag Detection)

ระบบตรวจจับ flag อัตโนมัติจากทั้งข้อความที่ LLM พิมพ์ตอบ และผลลัพธ์ดิบจาก tool (เผื่อ LLM ไม่ยอมพูดซ้ำ flag ที่เจอ) ด้วย regex pattern:

```
(?:flag|FLAG|HTB|THM|CTF|picoCTF|HackTheBox)\{[^\}\s]+\}
\b[a-f0-9]{32}\b   # hex 32 ตัวอักษร (รูปแบบ flag ของ HTB user/root)
```

6.3 จุดเด่นทางวิศวกรรม

- **Persistent HTTP session + auto-CSRF replay** — cookie อยู่ตลอด session, ดึง/ใส่ CSRF token ให้อัตโนมัติ
- **Loop detection** — ถ้า agent เรียก tool ซ้ำเดิม 3 ครั้งติด จะบังคับให้ agent วางแผนใหม่
- **Prompt caching** สำหรับโมเดล Anthropic เพื่อลดต้นทุน
- **Trust boundary** — ผลลัพธ์จาก tool ถูกห่อเป็น `<UNTRUSTED_DATA>` เพื่อป้องกัน prompt injection ที่ซ่อนอยู่ในหน้าเว็บเป้าหมาย
- รองรับ **custom exploit script** ตาม CVE-ID จากโฟลเดอร์ `custom_exploits/`

บทที่ 7 — ระบบที่ 3: Frontend (SecureThai Dashboard)

แดชบอร์ดเว็บที่สร้างด้วย Next.js 16 + HeroUI v3 ทำหน้าที่เป็นหน้าตาให้ผู้ใช้ทั่วไปสั่งงาน/ดูผลการสแกนโดยไม่ต้องใช้ command line

7.1 โครงสร้างหน้าจอ

Component/View	หน้าที่
`AppShell`	โครงหลัก: Sidebar + Topbar + ตัว route มุมมอง
`Sidebar`	เมนู Overview / Live Feed / Sessions / Findings / Memory / Settings พร้อมจุดสถานะ daemon และปุ่มเริ่ม scan ใหม่
`Topbar`	แสดงสถานะ scan ที่กำลังทำงาน + ปุ่มหยุด
`OverviewView`	ภาพรวมทั้งหมด
`LiveFeedView`	ดูความเคลื่อนไหวของ scan แบบ real-time ผ่าน SSE
`FindingsView`	รายการช่องโหว่ที่พบพร้อม verdict
`MemoryView`	ดูความจำข้าม-scan ต่อ host
`SettingsView`	ตั้งค่า API key/model

7.2 วิธีเชื่อมกับ Backend

แดชบอร์ดถูก build เป็น static export ไปอยู่ที่ `frontend/out` แล้วให้ backend เสิร์ฟไฟล์เอง — เท่ากับมี "1 process, 1 port (8000)" เดียวจบ ไม่ต้องกังวลเรื่อง CORS เพราะเรียก API แบบ same-origin ผ่าน `/api/...` ทุกครั้ง ไฟล์ `frontend/src/lib/api.ts` เขียน TypeScript interface ให้ตรงกับ Pydantic model ฝั่ง backend เป๊ะ ๆ เพื่อไม่ให้ type ไม่ตรงกัน

Token สำหรับยืนยันตัวตนถูกอ่านจาก URL fragment (`#token=...`) เท่านั้น ไม่เคยถูกส่งไปที่ server หรือถูกบันทึก log เพื่อความปลอดภัย

บทที่ 8 — ระบบที่ 4: Benchmark (XBOW Validation)

ใช้วัดความแม่นยำของ agent ก่อนส่งแข่งจริง โดยใช้ชุดโจทย์มาตรฐานอุตสาหกรรมชื่อ **XBOW validation benchmark**

รายการ	รายละเอียด
จำนวนโจทย์	104 โจทย์
ประเภทช่องโหว่ที่ครอบคลุม	15+ ประเภท เช่น XSS (27), IDOR (16), default credentials (19), privilege escalation (14), SSTI (14), command injection (12), business logic (7), SQLi (6), insecure deserialization (6), LFI (6), CVE (5), JWT (3), SSRF (3), race condition (1), HTTP smuggling (1)
วิธีรัน	`python3 run_benchmarks.py --all --pattern-flag` (ดูตัวเลือกเพิ่มเติมใน `benchmark/standalone-xbow-benchmark-runner/README.md`)
ที่มาโจทย์	เป็น git submodule ชี้ไปที่ repository `xbow-validation-benchmarks` — แต่ละโจทย์รันเป็น Docker container แยก

บทที่ 9 — การทำงานแบบ End-to-End (Scan Lifecycle)

```
ผู้ใช้กด "Start Scan" ในแดชบอร์ด (ใส่ URL เป้าหมาย + config)
   |
   ▼
POST /api/scans  →  ตรวจสิทธิ์ (authz) → เลือก Mode → สร้าง SafetyPolicy → คืน scan_id ทันที
   |
   ▼
[ Pre-scan Phase — ทำงานเบื้องหลังก่อน AI เริ่ม ]
   Phase 0a: หา subdomain (ถ้าเปิดใช้)
   Phase 0b: ตรวจจับ WAF (fingerprint)
   Phase 0c: Spider crawl หา endpoint/form/JS API แบบขนาน
   |
   ▼
[ ReAct Agent Loop — Orchestrator.run() ]
   LLM คิด → เรียก tool (http_request / dir_bruteforce) → ตรวจ SafetyPolicy ก่อนยิงจริง
   → เก็บผลลง Evidence Store → LLM รายงาน finding (อ้าง request_id) → วนซ้ำจนครบ/หมด budget
   |
   ▼
[ Verification Phase ]
   Oracle (IdorOracle/GeneralOracle) ดึงหลักฐานจริงจาก Evidence Store มาตรวจ pattern
   → ตัดสิน verdict: confirmed / likely / inconclusive / refuted
   |
   ▼
[ Persistence ]
   บันทึกผลลง PostgreSQL (ถ้าตั้งค่าไว้) + อัปเดตความจำข้าม-scan ของ host นี้
   |
   ▼
GET /api/scans/{id}/stream  →  ผู้ใช้เห็นผลแบบ real-time ผ่าน SSE ตลอดกระบวนการ
GET /api/scans/{id}/report  →  ดาวน์โหลดรายงาน HTML/PDF เมื่อเสร็จ
```

บทที่ 10 — กลไกความปลอดภัยของระบบเอง

เพราะ ShadowCat คือเครื่องมือที่ "โจมตี" เว็บอื่นโดยอัตโนมัติ ตัวระบบเองจึงต้องมีรั้วความปลอดภัยหลายชั้น เพื่อไม่ให้กลายเป็นเครื่องมืออันตรายที่ควบคุมไม่ได้:

10.1 SafetyPolicy — คุมขอบเขตทุก request

ทุกครั้งที่ agent จะเรียก tool ต้องผ่านจุดตรวจเดียว (`Orchestrator.dispatch_tool`) เสมอ ตรวจ 2 เรื่อง: (1) URL/host ปลายทางอยู่ใน scope ที่อนุญาตไหม (2) HTTP method ที่ใช้ อนุญาตไหม (เช่น enterprise mode ห้าม POST/DELETE) ถ้าผิดเงื่อนไข tool จะ**ไม่ถูกเรียกจริงเลย** — agent ได้แค่ error message กลับไป

10.2 ป้องกัน SSRF ไปยัง Cloud Metadata

บล็อก host กลุ่ม cloud metadata (เช่น `169.254.169.254`, `metadata.google.internal`) แบบตายตัว แม้ scope ที่ตั้งไว้จะครอบคลุมก็ตาม เพื่อกันไม่ให้ agent หลุดไปดึงข้อมูล credential ของ cloud provider เข้ามาโดยไม่ตั้งใจ

10.3 Credential Isolation

Agent (LLM) **ไม่มีทางเห็น** username/password/cookie จริงเลย — ถ้า LLM พยายามใส่ header `Authorization` / `Cookie` เอง ระบบจะตัดทิ้งทันทีแล้ว inject ค่าจริงจากฝั่งเซิร์ฟเวอร์แทน สิ่งที่ agent เห็นกลับมามีแค่ "ชื่อ identity" เช่น `userA` ไม่ใช่ตัว secret

10.4 Evidence Grounding — ป้องกัน AI มั่ว (Anti-Hallucination)

1. LLM เห็นแค่ตัวอย่าง response สั้น ๆ ไม่เห็นเต็ม
2. เวลารายงาน finding ต้องอ้าง request_id ที่มีอยู่จริงใน Evidence Store เท่านั้น ถ้าอ้าง ID ที่ไม่มีจริง ระบบปฏิเสธการรายงานทันที
3. โครงสร้างข้อมูลของการรายงาน (schema) **ไม่มีช่องให้ LLM ใส่ severity/status เอง** — ป้องกันไม่ให้ agent "ตัดสินตัวเอง"
4. Oracle ตัดสินแยกจาก LLM 100% โดยใช้กฎตายตัว ไม่เรียก LLM มาช่วยตัดสินซ้ำ
5. System prompt ย้ำอีกชั้นว่า "ห้ามสรุปเองว่าเจอช่องโหว่ ถ้าไม่ได้ยิง request จริง ถือว่าไม่เกิดขึ้น"

10.5 Authorization Gate + Audit Log

แยกจาก SafetyPolicy อีกชั้นหนึ่ง — ก่อนจะ "เริ่ม" scan ได้เลย ระบบตรวจ target กับตัวแปร environment `SHADOWCAT_AUTHORIZED_SCOPE` ถ้าตั้งค่าไว้และ target ไม่อยู่ในนั้น จะถูกปฏิเสธด้วย 403 ทันที ทุกความพยายาม (อนุญาตหรือปฏิเสธ) จะถูกบันทึกเป็น JSON log ที่ `~/.shadowcat/audit.log` เพื่อตรวจสอบย้อนหลังได้

10.6 สรุปกลไกทั้งหมดเป็นตาราง

ชั้นความปลอดภัย	ป้องกันอะไร
SafetyPolicy (scope + method check)	agent หลุด scope หรือใช้ method ที่ไม่ควร (เช่น DELETE โดยไม่ตั้งใจ)
Blocked metadata hosts	SSRF ไปขโมย credential จาก cloud metadata service
Credential isolation	credential รั่วไหลผ่าน LLM หรือถูก LLM override
Evidence grounding + schema บังคับ	AI hallucination / รายงานช่องโหว่ปลอม
Oracle แยกจาก LLM	ผลตัดสินไม่สม่ำเสมอ (inconsistent) จาก AI
Authorization gate + audit log	การสแกนเป้าหมายที่ไม่ได้รับอนุญาต / ไม่มีหลักฐานย้อนสอบ

บทที่ 11 — การทดสอบระบบ (Testing)

ใช้ pytest + pytest-asyncio ครอบคลุมทั้ง backend และ agent

กลุ่มเทสต์	ครอบคลุมอะไร
`tests/test_enterprise_api.py`	การบังคับ scope, รูปแบบ SSE event, ผลตัดสินของ oracle, endpoint ทั้ง 5 ของ enterprise mode
`tests/test_enterprise_engine.py`	credential isolation, evidence grounding, ความเข้มงวดของ report_finding
`tests/test_general_oracle.py`	ความถูกต้องของ verdict แบบมีลำดับชั้น (confirmed vs likely)
`tests/test_daemon_cli.py`	สถานะ daemon, self-heal, authz allow/deny, audit log
`tests/test_scan_memory.py`	ความถูกต้องของระบบความจำข้าม scan
`tests/test_user_config.py`	การอ่าน/เขียนค่าตั้งค่าผู้ใช้ผ่าน API
`tests/unit/, tests/integration/, tests/docker/`	ทดสอบ package agent/ แยกต่างหาก (event bus, flag detection, session, controller, docker build)

คำสั่งหลัก: `make test` (รันทุกเทสต์ ยกเว้น docker), `make test-cov` (พร้อม coverage), `make ci` (จำลอง CI เต็มรูปแบบ: lint → format check → typecheck → test → build)

สถานะล่าสุดที่บันทึกไว้ใน PROGRESS.md: 265 เทสต์ผ่าน / 27 skip, ruff + mypy สะอาด

บทที่ 12 — คู่มือการนำเสนอ

12.1 สคริปต์นำเสนอ 3 นาที

เปิดเรื่อง (30 วินาที) — เริ่มด้วยปัญหา ไม่ใช่เทคนิค

> "สวัสดีครับ ทีม ShadowCat ครับ
>
> ปัญหา: การทดสอบความปลอดภัยเว็บ (Pentest) มีค่าใช้จ่ายสูง — จ้างบริษัทนอก 50,000–200,000 บาทต่อครั้ง ใช้เวลา 1–2 สัปดาห์ และขาดแคลนผู้เชี่ยวชาญ
>
> ทางแก้: ShadowCat — AI Pentest Agent ที่ทำงานอัตโนมัติ ค่าใช้จ่ายหลักร้อยบาทต่อครั้ง ใช้เวลา 5–10 นาที ใช้งานได้ทันที"

สาธิตสด (1.5 นาที)

> 1. เปิดแดชบอร์ด ใส่ URL เป้าหมาย → กด Start
> 2. ระบบจะ: ตรวจจับ WAF → Crawl หา endpoint/form → ทดสอบ OWASP Top 10 → เก็บหลักฐานทุก request/response
> 3. ผลลัพธ์: แสดง findings พร้อมระดับความรุนแรง แต่ละ finding มีหลักฐาน (request_id) และ Oracle ยืนยันว่า "confirmed" หรือ "likely"
> 4. ชี้จุดเด่น: Memory (จำผลจาก scan ก่อนหน้า), Cost Control (ตั้ง budget ได้), Report (ดาวน์โหลด HTML/PDF)

ปิดเรื่อง (1 นาที) — ย้ำจุดขาย 4 ข้อ

> 1. **Evidence-based Verification** — AI ไม่ตัดสินใจเอง, Oracle ตรวจสอบอีกที ไม่มี false positive จาก AI hallucination
> 2. **Deterministic Oracle** — ตรวจจาก pattern จริง (SQL error, /etc/passwd, private key)
> 3. **Memory System** — จำ endpoint/finding จาก scan ก่อน scan ครั้งที่ 2 เร็วกว่า 30–50%
> 4. **ราคา** — $1–2/scan เทียบกับ manual pentest $5,000+ ถูกกว่า 100 เท่า
>
> ขอบคุณครับ

12.2 โครงสไลด์ (10 สไลด์)

#	หัวข้อ	เนื้อหาหลัก
1	Title	ShadowCat — AI-Powered Autonomous Penetration Testing Agent — NSC 2026

2	Problem	ค่าใช้จ่ายสูง / ใช้เวลานาน / ขาดผู้เชี่ยวชาญ / ทำไม่บ่อยพอ
3	Solution	ค่าใช้จ่ายหลักร้อย / 5-10 นาที / ใช้งานได้ทันที / ทำได้บ่อยขึ้น
4	Architecture	แผนภาพ: User Input → Pre-Scan (Subdomain/WAF/Spider) → ReAct Loop → Oracle → Report
5	Key Features	Evidence-based, Memory, Cost Control, OWASP Top 10 Coverage
6	Demo	สาธิตสด: ใส่ URL → รอผล → ดู findings → ดาวน์โหลด report
7	Benchmark Results	XBOW 104 tests: L1 91.1% / L2 74.5% / L3 62.5% / รวม 86.5%
8	Comparison	เทียบ ShadowCat vs Burp Suite vs OWASP ZAP (ราคา/ความง่าย/AI-driven/Auto-verify/Memory)
9	Use Cases	Developer / Security Team / Compliance / Bug Bounty
10	Thank You	ช่องทางติดต่อ + เปิดรับคำถาม

12.3 เทคนิคการนำเสนอ

เทคนิค	รายละเอียด
เปิดด้วยปัญหา	อย่าเริ่มด้วย technical details เริ่มด้วยปัญหาที่กรรมการ relate ได้ก่อน
สาธิตสด	เตรียม demo script ที่ทำงานได้เสมอ + มี video/screenshot สำรองถ้า demo ล่ม
ใช้ตัวเลขที่จับต้องได้	"ถูกกว่า 100 เท่า" ดีกว่า "ถูกกว่ามาก", "86.5% success rate" ดีกว่า "ทำงานได้ดี"
ตอบคำถามมีโครงสร้าง	Problem → Solution → Benefit อย่าตอบวนไปมา
ยอมรับถ้าไม่รู้	"คำถามดีครับ ขอจดไปวิจัยเพิ่ม" ดีกว่าตอบมั่ว

12.4 Quick Tips ก่อนขึ้นเวที

- ฝึกพูดอย่างน้อย 3 รอบก่อนนำเสนอจริง
- จับเวลาให้ได้ 3 นาทีพอดี
- เตรียม backup (วิดีโอ, ภาพหน้าจอ) เผื่อ demo ล่ม
- นอนให้พอคืนก่อนแข่ง และไปถึงสถานที่แต่เช้าเพื่อเตรียมอุปกรณ์

12.5 ถ้าลืม/ประหม่าตอนนำเสนอ

1. หยุดหายใจลึก ๆ 3 ครั้ง
2. มองสไลด์แล้วพูดตามโครงที่เตรียมไว้
3. พูดช้าลงเมื่อรู้สึกประหม่า
4. ยอมรับตรง ๆ ได้ เช่น "ขอโทษครับ ขอนึกสักครู่"

บทที่ 13 — คู่มือการตอบคำถามกรรมการ + ตัวอย่างคำถาม

แต่ละคำถามด้านล่างเป็นคำถามที่คาดว่ากรรมการมักถามจริง พร้อมแนวคำตอบสั้น กระชับ ใช้ตัวเลข/ข้อเท็จจริงจากตัวระบบจริง

Q1: ต่างจาก Burp Suite / OWASP ZAP อย่างไร?
Burp/ZAP เป็นเครื่องมือที่ต้องมีคนใช้และมีความรู้ pentest ส่วน ShadowCat เป็น AI Agent ที่ทำงานอัตโนมัติเต็มรูปแบบ — ใส่แค่ URL ระบบจะเลือกทดสอบจุดสำคัญเอง แล้วตรวจผลด้วย Oracle เพื่อไม่ให้ miss หรือมั่วผล ราคาถูกกว่ากันมาก ($1 เทียบ $5,000+)

Q2: AI hallucination แก้ปัญหาอย่างไร?
แยกหน้าที่ชัดเจน: LLM ทำหน้าที่ "เก็บหลักฐาน" เท่านั้น (ส่ง request → เก็บ response) ไม่ตัดสินว่าใช่ช่องโหว่หรือไม่ ส่วน Oracle (rule-based) เป็นผู้ "ตัดสิน" จากการตรวจ pattern เช่น SQL error, เนื้อหา /etc/passwd, XSS ที่ reflect กลับมาจริง ผลลัพธ์แบ่งเป็น confirmed / likely / inconclusive

Q3: WAF bypass ทำอย่างไร?
ขั้นแรกตรวจจับ WAF ก่อนด้วยการยิง probe แล้วดู header/body signature (เช่น cf-ray ของ Cloudflare) ส่วนเทคนิค evasion (URL encoding, double encoding, unicode escape, case variation, SQL comment, rotating headers) มีโค้ดเตรียมไว้แล้วแต่**ยังไม่ได้เชื่อมเข้ากับ orchestrator จริง** — อยู่ในแผนพัฒนาต่อ (ควรตอบตรงไปตรงมาแบบนี้)

Q4: Memory ทำงานอย่างไร?
หลัง scan เสร็จ ระบบบันทึกข้อมูล (endpoint ที่ทดสอบแล้ว, finding ที่พบ, WAF/เทคโนโลยีที่เจอ) ลงไฟล์ JSON แยกตาม host ก่อน scan ครั้งถัดไปในโดเมนเดียวกัน agent จะโหลดข้อมูลนี้กลับมาก่อน ทำให้ไม่ต้องทดสอบซ้ำจุดเดิม เร็วขึ้นประมาณ 30–50% และประหยัดค่าใช้จ่าย

Q5: Benchmark น่าเชื่อถือแค่ไหน?
ใช้ XBOW validation benchmark suite ซึ่งเป็นมาตรฐานที่ยอมรับในอุตสาหกรรม จำนวน 104 โจทย์ แบ่ง 3 ระดับความยาก ผลรวมอยู่ที่ 86.5% success rate เฉลี่ย $1.11 ต่อ scan

Q6: ถ้าเจอช่องโหว่ที่ Oracle ไม่มี pattern รองรับล่ะ?
Oracle ปัจจุบันครอบคลุม OWASP Top 10 หลัก ๆ (SQLi, XSS, Path Traversal, IDOR, SSRF, Open Redirect, Auth Bypass, Sensitive Data) ถ้าเจอช่องโหว่ที่ไม่มี pattern รองรับ ระบบจะรายงานเป็น "inconclusive" แทนที่จะฟันธงมั่ว ๆ และนักพัฒนาสามารถเพิ่ม pattern ใหม่เข้า Oracle ได้ภายหลัง โดยไม่ต้อง train AI ใหม่

Q7: ใช้ LLM ตัวไหน ทำไม?
รองรับหลาย provider: PSU Blue (ถูก/เร็ว), OpenRouter (เลือกโมเดลได้หลากหลาย), Anthropic Claude

โดยตรง (คุณภาพสูง), และ local LLM (ฟรี ไม่เสียเงิน) แนะนำ OpenRouter + โมเดลตระกูล coder ราคาประหยัด เพราะรองรับ function calling ได้ดีและคุณภาพเพียงพอสำหรับงาน pentest

Q8: ตัวระบบเองปลอดภัยแค่ไหน?
มี 4 ชั้น: (1) SafetyPolicy ตรวจ scope/method ทุก request (2) Credential isolation — agent ไม่เห็น credential จริง (3) Audit log บันทึกทุกความพยายาม scan แบบ JSONL (4) Authorization gate ผ่าน environment variable กำหนดขอบเขตที่อนุญาตล่วงหน้า

Q9: แผนพัฒนาต่อไปคืออะไร?
เชื่อม WAF evasion เข้า orchestrator จริง, ทำ independent oracle sweep (ให้ Oracle ตรวจ evidence store ทั้งหมดเองโดยไม่พึ่ง agent รายงาน), รองรับ multi-identity spider (crawl พร้อมกันหลาย identity เพื่อจับ IDOR ได้มากขึ้น), และขยายไปทดสอบ mobile API/GraphQL

Q10: ทำไมใช้ Python ไม่ใช้ Go/Rust?
เพราะ ecosystem ของ LLM (OpenAI/Anthropic SDK, เครื่องมือ crawl อย่าง httpx/BeautifulSoup/Playwright) เป็น Python เกือบทั้งหมด, FastAPI เร็วพอสำหรับ use case นี้และรองรับ async ดี, ทำให้พัฒนาได้เร็วกว่าภายในเวลาจำกัดของการแข่งขัน

Q11: ReAct ต่างจาก Chain-of-Thought อย่างไร?
Chain-of-Thought คือ "คิด → คิด → คิด → ตอบ" ไม่มี action จริงเกิดขึ้น ส่วน ReAct คือ "คิด → ทำ (เรียก tool จริง) → ดูผลจริง → คิดต่อ" วนซ้ำ ทำให้ตัดสินใจจากข้อมูลจริงที่เพิ่งได้มา ไม่ใช่แค่คาดเดาในหัว

Q12: Memory ต่างจาก Cache อย่างไร?
Cache เก็บข้อมูลชั่วคราว ลบเมื่อไหร่ก็ได้ ไม่มีโครงสร้าง ส่วน Memory ของ ShadowCat เก็บถาวรเป็นไฟล์ JSON มีโครงสร้างชัดเจน (endpoints/findings/tech/waf) และสะสมข้อมูลเพิ่มขึ้นเรื่อย ๆ ข้าม session ได้

Q13: ทำไมใช้ BFS ไม่ใช้ DFS ตอน crawl เว็บ?
BFS (Breadth-First) จะสำรวจความลึกระดับ 1 ให้ครบก่อนค่อยลึกลงไประดับถัดไป ทำให้เจอหน้าสำคัญ (login, admin, api) ที่มักอยู่ใกล้หน้าแรกได้เร็วกว่า ครอบคลุมกว่า ไม่ติดหลงอยู่ใน branch เดียว และควบคุม max_depth ได้ง่าย ต่างจาก DFS ที่อาจหลงลึกไปในกิ่งที่ไม่สำคัญจนพลาดหน้าหลัก

Q14: ค่าใช้จ่ายของ LLM ควบคุมอย่างไร?
4 เทคนิคหลัก: (1) History Pruning — ตัดประวัติบทสนทนาเก่าเป็นระยะ (2) Body Preview Limit — จำกัดเนื้อหา response ที่ส่งให้ agent เหลือ ~900 ตัวอักษร (ลด token ~90%) (3) Concurrent Tool Dispatch — เรียกหลาย tool พร้อมกันใน 1 turn แทนที่จะเรียกทีละตัว (4) Coverage Ledger — เตือนไม่ให้ agent ทดสอบจุดเดิมซ้ำทุก 6 turn

Q15: victim_markers คืออะไร ทำไมต้องมี?
เป็นข้อมูลที่ผู้ควบคุมสแกนใส่เข้ามาล่วงหน้า (เช่น อีเมลของผู้ใช้ B) เพื่อให้ Oracle พิสูจน์ IDOR ได้อย่างแม่นยำว่า response ที่ผู้ใช้ A เข้าถึงได้นั้น "เป็นข้อมูลของ B จริง" ไม่ใช่แค่บังเอิญ status code เป็น 200 — ถ้าไม่มี marker เหล่านี้ ระบบจะให้ผลแค่ "inconclusive" เท่านั้น ไม่ฟันธงว่า confirmed

13.1 แม่แบบการตอบเมื่อไม่รู้คำตอบ

"คำถามดีครับ ขอนึกสักครู่..." (หยุด 2-3 วินาที) แล้วตอบเท่าที่รู้จริง ปิดท้ายด้วย "รายละเอียดส่วนนี้ขอจดไปศึกษาเพิ่มเติมครับ"

13.2 แม่แบบการตอบเมื่อกรรมการจับผิดจุดอ่อน

"ขอบคุณครับที่ชี้จุดนี้ ระบบยังมีข้อจำกัดตรงนี้จริง [ยอมรับตรง ๆ] และอยู่ในแผนพัฒนาที่จะแก้ไขต่อไปครับ [บอกแผน]" — การยอมรับตรง ๆ น่าเชื่อถือกว่าการเถียงหรือกลบเกลื่อน

บทที่ 14 — คำศัพท์ทั้งหมดที่ควรรู้

รวบรวมจากเอกสารเทคนิคของโปรเจค จัดกลุ่มตามหมวดเพื่อให้ทบทวนง่าย มีคำอ่านภาษาไทยกำกับสำหรับคำที่ออกเสียงยาก

14.1 หมวด AI / Agent / LLM

คำศัพท์	คำอ่าน	ความหมาย
Agent	เอ-เจ้นต์	ตัวแทน AI ที่ทำงานอัตโนมัติแทนมนุษย์
LLM	แอล-แอล-เอ็ม	Large Language Model โมเดลภาษาขนาดใหญ่ที่ขับเคลื่อน agent
ReAct	รี-แอคต์	Reasoning + Acting รูปแบบที่ AI คิด-ทำ-ดูผล-คิดต่อ วนซ้ำ
Reasoning	รี-ซัน-นิ่ง	การให้เหตุผล/คิดว่าควรทำอะไรต่อ
Action	แอค-ชั่น	การกระทำ/การเรียก tool ของ agent
Observation	ออบ-เซอร์-เว-ชั่น	การสังเกตผลลัพธ์ที่ได้กลับมาหลัง action
Prompt	พรอมต์	คำสั่ง/ข้อความที่ป้อนให้ LLM
System Prompt	ซิส-เต็ม พรอมต์	คำสั่งตั้งต้นที่กำหนดบทบาทและกฎของ agent
Token	โท-เคิน	หน่วยข้อความย่อยที่ LLM ใช้คิดค่าใช้จ่าย
Hallucination	ฮา-ลู-ซิ-เน-ชั่น	อาการ AI "มั่ว" สร้างข้อมูลที่ไม่เป็นจริงขึ้นมาเอง
Function Calling / Tool Calling	ฟังก์ชัน คอลลิ่ง	ความสามารถของ LLM ในการเรียกใช้ฟังก์ชัน/เครื่องมือภายนอก
Planner	แพลน-เนอร์	โมดูลที่ให้ LLM วางแผนล่วงหน้าก่อนเริ่ม loop หลัก
Context Pruning	คอน-เท็กซ์ พรูน-นิ่ง	การตัดประวัติบทสนทนาเก่าทิ้งเพื่อลดต้นทุน
Prompt Caching	พรอมต์ แคช-ชิ่ง	การแคชส่วนของ prompt ที่ไม่เปลี่ยนเพื่อลดค่าใช้จ่ายซ้ำ
Prompt Injection	พรอมต์ อิน-เจค-ชั่น	การแอบฝังคำสั่งในข้อมูลที่ agent อ่าน เพื่อหลอกให้ agent ทำตาม

14.2 หมวดการตรวจสอบ/พิสูจน์ผล (Verification)

คำศัพท์	คำอ่าน	ความหมาย
Oracle	ออ-รา-เคิล	โมดูล rule-based ที่ทำหน้าที่ตัดสินผล ไม่ใช่ AI
Evidence	เอ-วิ-เดินส์	หลักฐาน (คู่ request/response จริง)
Evidence Store	เอ-วิ-เดินส์ สโตร์	ที่เก็บหลักฐานทุกคู่ request/response แบบเพิ่มได้อย่างเดียว
Finding	ไฟน์-ดิ้ง	สิ่งที่ agent สงสัยว่าเป็นช่องโหว่ ยังไม่ผ่านการยืนยัน
Verdict	เวอร์-ดิกต์	คำตัดสินขั้นสุดท้ายจาก Oracle

Confirmed	คอน-เฟิร์ม	ยืนยันแล้วว่าเป็นช่องโหว่จริง
Likely	ไลค์-ลี่	น่าจะเป็นช่องโหว่ แต่ยังไม่ฟันธง
Inconclusive	อิน-คอน-คลู-ซีฟ	ข้อมูลไม่พอสรุปผล
Refuted	รี-ฟิวต์-เต็ด	พิสูจน์แล้วว่าไม่ใช่ช่องโหว่
Pattern Matching	แพท-เทิร์น แมช-ชิ่ง	การเทียบข้อมูลกับรูปแบบที่กำหนดไว้ตายตัว
Regex	รี-เจ็กซ์	Regular Expression ภาษาสำหรับกำหนดรูปแบบข้อความ
Victim Markers	วิค-ทิม มาร์กเกอร์	ข้อมูลระบุตัวตนของเหยื่อที่ใส่ล่วงหน้า เพื่อพิสูจน์ IDOR

14.3 หมวดประเภทช่องโหว่ (Vulnerability Types)

คำศัพท์	คำอ่าน	ความหมาย
Vulnerability	วัล-เน-รา-บิ-ลิ-ตี้	ช่องโหว่ความปลอดภัย
Exploit	เอ็กซ์-พลอยต์	การใช้ประโยชน์จากช่องโหว่เพื่อโจมตี
SQLi (SQL Injection)	เอส-คิว-แอล-ไอ	การสอดแทรกคำสั่ง SQL ผ่าน input ของผู้ใช้
XSS	เอ็กซ์-เอส-เอส	Cross-Site Scripting การสอดแทรกสคริปต์อันตรายในหน้าเว็บ
IDOR	ไอ-ดอ	Insecure Direct Object Reference การเข้าถึงข้อมูลของผู้ใช้อื่นผ่านการเปลี่ยนค่า reference
SSRF	เอส-เอส-อาร์-เอฟ	Server-Side Request Forgery การหลอกให้เซิร์ฟเวอร์ยิง request แทนผู้โจมตี
LFI / Path Traversal	แอล-เอฟ-ไอ	Local File Inclusion การเข้าถึงไฟล์ในระบบที่ไม่ควรเข้าถึงได้
RCE	อาร์-ซี-อี	Remote Code Execution การรันโค้ดบนเซิร์ฟเวอร์เป้าหมายจากระยะไกล
SSTI	เอส-เอส-ที-ไอ	Server-Side Template Injection การสอดแทรกโค้ดผ่าน template engine
CSRF	ซี-เอส-อาร์-เอฟ	Cross-Site Request Forgery การหลอกให้ผู้ใช้ทำ request โดยไม่ตั้งใจ
Open Redirect	โอเพ่น รี-ไดเรกต์	ช่องโหว่ที่ทำให้เว็บ redirect ไปปลายทางอันตรายได้
Auth Bypass	ออธ ไบ-พาส	การข้ามระบบยืนยันตัวตนโดยไม่ได้รับอนุญาต
CWE	ซี-ดับเบิลยู-อี	Common Weakness Enumeration รหัสมาตรฐานจำแนกประเภทช่องโหว่
OWASP Top 10	โอ-แวสป์	รายการช่องโหว่เว็บที่พบบ่อยที่สุด 10 อันดับ จัดโดยองค์กร OWASP
Severity	เซ-เว-ริ-ตี้	ระดับความรุนแรงของช่องโหว่ (Critical/High/Medium/Low/Info)

14.4 หมวดเว็บ/เครือข่าย (Web & Network)

คำศัพท์	คำอ่าน	ความหมาย
HTTP / HTTPS	เอช-ที-ที-พี	โปรโตคอลสื่อสารของเว็บ (HTTPS คือเวอร์ชันเข้ารหัส)
Endpoint	เอนด์-พอยต์	จุดปลายทาง/URL ที่เรียกใช้งานได้
API	เอ-พี-ไอ	Application Programming Interface ช่องทางให้โปรแกรมคุยกัน
REST	เรสต์	รูปแบบการออกแบบ API ที่นิยมใช้บน HTTP
Payload	เพ-โหลด	ข้อมูลที่ส่งไปทดสอบ/โจมตีเป้าหมาย
Cookie / Session	คุก-กี้ / เซส-ชั่น	ข้อมูลที่ใช้จำสถานะการล็อกอินของผู้ใช้
Header / Body	เฮด-เดอร์ / บอ-ดี้	ส่วนหัว และเนื้อหาของ HTTP request/response
WAF	ดับเบิลยู-เอ-เอฟ	Web Application Firewall ระบบป้องกันหน้าเว็บจากการโจมตี
Rate Limit	เรท ลิ-มิต	การจำกัดจำนวน request ในช่วงเวลาหนึ่ง
DNS	ดี-เอ็น-เอส	Domain Name System ระบบแปลงชื่อโดเมนเป็น IP
Subdomain	ซับ-โด-เมน	โดเมนย่อยของโดเมนหลัก
CIDR	ซิ-เดอร์	รูปแบบระบุช่วง IP address
SPA	เอส-พี-เอ	Single Page Application เว็บที่ render ด้วย JavaScript ฝั่ง client

14.5 หมวด Crawler / Spider

คำศัพท์	คำอ่าน	ความหมาย
Spider / Crawl	สไป-เดอร์ / ครอว์ล	โปรแกรมที่สำรวจเว็บอัตโนมัติเพื่อหาหน้า/endpoint ทั้งหมด
BFS	บี-เอฟ-เอส	Breadth-First Search สำรวจทีละระดับความลึกให้ครบก่อน
DFS	ดี-เอฟ-เอส	Depth-First Search สำรวจลึกไปเรื่อย ๆ ก่อนถอยกลับ
Headless Browser	เฮด-เลส เบราว์เซอร์	เบราว์เซอร์ที่รันโดยไม่มีหน้าจอ ใช้ render JS อัตโนมัติ
Playwright / Chromium	เพลย์-ไรท์ / โคร-เมียม	เครื่องมือ/เอนจินสำหรับควบคุมเบราว์เซอร์อัตโนมัติ
Fingerprint	ฟิง-เกอร์-พริ้นท์	การระบุลักษณะเฉพาะเพื่อจำแนกประเภท (เช่น ยี่ห้อ WAF)

14.6 หมวดความปลอดภัยของระบบเอง (Safety Engineering)

คำศัพท์	คำอ่าน	ความหมาย
Scope	สโคป	ขอบเขตเป้าหมายที่อนุญาตให้ทดสอบได้
Policy	พอ-ลิ-ซี่	นโยบาย/กฎที่ระบบต้องปฏิบัติตาม
SafetyPolicy	เซฟ-ตี้ พอ-ลิ-ซี่	โมดูลตรวจสอบ scope และ method ก่อน agent ยิง request จริง
Credential	ครี-เดน-เชียล	ข้อมูลยืนยันตัวตน เช่น username/password

Identity	ไอ-เดน-ติ-ตี้	ตัวตนผู้ใช้ที่ระบบจัดการแทน agent (เช่น userA, userB)
Authentication	ออ-เธน-ติ-เค-ชั่น	การยืนยันตัวตนว่าเป็นใคร
Authorization	ออ-ธอ-ไร-เซ-ชั่น	การกำหนดสิทธิ์ว่าทำอะไรได้บ้าง
Audit Log	ออ-ดิท ล็อก	บันทึกเหตุการณ์เพื่อการตรวจสอบย้อนหลัง
Evasion	อี-เว-ชั่น	การหลบเลี่ยงระบบป้องกัน เช่น WAF
Encoding	เอน-โคด-ดิ้ง	การแปลงรูปแบบข้อมูล มักใช้เพื่อหลบการตรวจจับ

14.7 หมวดระบบ/วิศวกรรมซอฟต์แวร์ (Systems & Engineering)

คำศัพท์	คำอ่าน	ความหมาย
Daemon	ดี-มอน	โปรแกรมที่รันอยู่เบื้องหลังตลอดเวลา
CLI	ซี-แอล-ไอ	Command Line Interface การสั่งงานผ่านบรรทัดคำสั่ง
SSE	เอส-เอส-อี	Server-Sent Events วิธีส่งข้อมูลจาก server ไป client แบบ real-time ทางเดียว
Async / Concurrent	เอ-ซิงค์ / คอน-เคอร์-เร้นท์	การทำงานแบบไม่รอ / การทำงานพร้อมกันหลายอย่าง
Docker / Container	ดอค-เกอร์ / คอน-เทน-เนอร์	เทคโนโลยีบรรจุโปรแกรมให้รันได้ทุกที่เหมือนกัน
CI/CD	ซี-ไอ-ซี-ดี	Continuous Integration/Delivery กระบวนการทดสอบ/ส่งมอบโค้ดอัตโนมัติ
Coverage Ledger	คัฟ-เวอร์-เรจ เลด-เจอร์	สมุดบันทึกภายในที่กันไม่ให้ agent ทดสอบจุดเดิมซ้ำ
Budget / Cost Control	บัด-เจท	การกำหนด/ควบคุมงบประมาณค่าใช้จ่ายของการสแกน
DAST	แดสท์	Dynamic Application Security Testing การทดสอบความปลอดภัยขณะแอปทำงานจริง
PDPA	พี-ดี-พี-เอ	พ.ร.บ. คุ้มครองข้อมูลส่วนบุคคลของไทย

ภาคผนวก — คำสั่งที่ใช้บ่อย

A.1 พัฒนาและรัน

```
uv sync                              # ติดตั้ง dependency ทั้งหมด
shadowcat up                         # เริ่ม daemon + เปิดแดชบอร์ด (Local-First Agent OS)
shadowcat status                     # ดูสถานะ daemon (pid, port, uptime)
shadowcat down                       # ปิด daemon
uv run uvicorn backend.api.routes_scan:app --port 8000   # รัน backend ตรง ๆ
uv run shadowcat-agent --target <IP/URL>                  # รัน agent CTF/HTB แบบ single-shot
```

A.2 Build หน้าเว็บแดชบอร์ด

```
cd frontend && npm install && npm run build && cd ..   # build → frontend/out ให้ backend เสิร์ฟ
cd frontend && NEXT_PUBLIC_API_URL=http://127.0.0.1:8000 npm run dev   # dev server แยก (hot reload)
```

A.3 ทดสอบและตรวจคุณภาพโค้ด

```
make test           # รันเทสต์ทั้งหมด (ยกเว้น docker)
make test-cov        # รันเทสต์พร้อม coverage
make lint             # ตรวจโค้ดด้วย ruff
make typecheck         # ตรวจ type ด้วย mypy
make ci                  # จำลอง CI เต็มรูปแบบ
```

A.4 Docker

```
make install         # build Docker image
make connect          # เข้าไปใช้งานในคอนเทนเนอร์
make stop              # หยุดคอนเทนเนอร์
```

A.5 Benchmark

```
cd benchmark/standalone-xbow-benchmark-runner
python3 run_benchmarks.py --range 1-10 --pattern-flag   # รันโจทย์ 1-10
python3 run_benchmarks.py --all --pattern-flag           # รันทั้งหมด 104 โจทย์
```