Portfolio - Marius Anderie

title: Code Intelligence & MCP Server summary: Built static-analysis tools and an MCP server that gave AI coding agents full codebase understanding — cutting task completion time by 4.7x. date: 2026-01 category: AI & Architecture tags: MCP, Python, TypeScript, C#, Roslyn, Docker

Code Intelligence & MCP Server for Legacy Enterprise App — Defense Client (Switzerland)

> ***Built static-analysis tools and an MCP server that gave AI coding agents full codebase understanding — cutting task completion time by 4.7x on a 500k+ LOC legacy system.***

- **Date:** Jan 2026 - Feb 2026
- **Roles:** AI/ML Engineer · AI Tooling Developer · Software Architect
- **Team size:** 1 (sole contributor)
- **Customer:** Confidential Defense Client (Switzerland)
- **Core Stack:**
  Frontend Analyzer: TypeScript · ts-morph · Angular Compiler · Node.js
  Backend Analyzer: C# · .NET · Roslyn (Microsoft.CodeAnalysis) · SQL Server
  MCP Server: Python 3.11 · MCP SDK · Pydantic · rapidfuzz · pyodbc · Docker
- **Blog post:** Building an MCP Server for Enterprise Codebases

1. Problem

The MILO5 enterprise portal (HR, finance, logistics, warehouse) is a large-scale Angular + .NET monolith serving 12 000 users. AI coding agents struggled with the codebase — they spent most of their time navigating files and guessing at architecture rather than solving problems. Additionally, the RAG chatbot's action data (see *Action-Tree* project) lacked backend context such as permissions, validations and service dependencies.

2. Solution Overview

I designed and built three tools that extract structured intelligence from both the frontend and backend, then serve it to AI agents via the Model Context Protocol:

Component	Purpose	Key Tech
Frontend Code Intelligence	Parses Angular templates, NgRx state management and API proxies to trace UI click → store dispatch → effect → backend call	TypeScript · ts-morph · Angular Compiler
Backend Code Intelligence	Analyzes controllers, services, authorization attributes, DB schema, enums and data-scoping rules via Roslyn	C# · .NET · Roslyn · SQL Server
MILO5 MCP Server	Exposes 6 tools (trace API flow, trace UI action, module context, reverse UI lookup, DB schema, read-only SQL queries) over MCP	Python · MCP SDK · rapidfuzz · Docker

3. Impact

- AI coding agents completed 5 benchmark tasks in **3 min** with MCP vs. **14 min** without — a **4.7x speedup**
- Simple bug fixes dropped from ~2 min to **20–30 s** per task
- Full-stack feature task dropped from 6 min to **2 min**
- Enriched all **715** chatbot actions with extracted business logic (preconditions, permissions, service dependencies, error scenarios) — the MCP-equipped agent analyzed the entire frontend and backend in **1.5 hours**
- Demonstrated a replicable approach for making any legacy codebase AI-ready

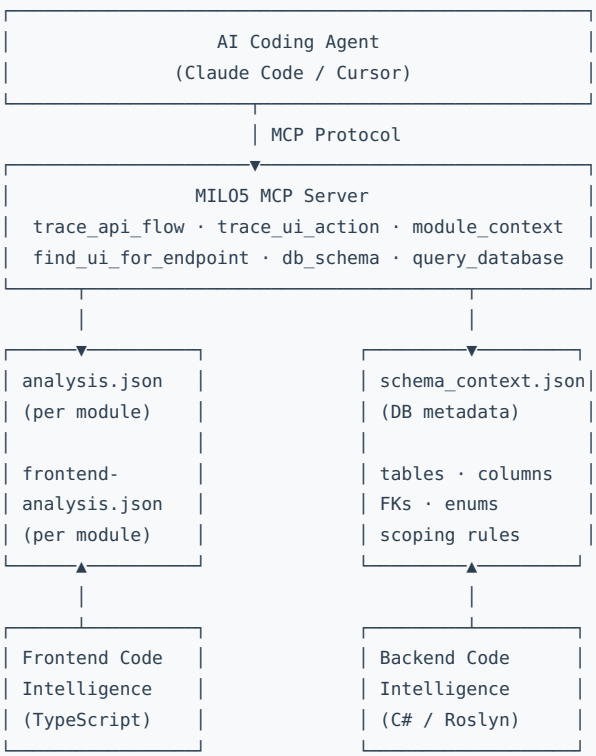
4. My Contributions

Designed the full code-intelligence architecture; *built* the TypeScript frontend analyzer (6 extractors + trace builder), the C# backend analyzer (Roslyn-based controller, auth, schema, enum and scoping analysis) and the Python MCP server; *defined* the 5-task evaluation benchmark from real commit history; *measured* before/after performance; *integrated* extracted business logic into the RAG chatbot's action data.

5. Key Challenges & Mitigations

Challenge	Mitigation & Result
Tracing UI events end-to-end through Angular + NgRx + API proxy layers	Built a TraceBuilder that indexes and links template events → component methods → dispatched actions → effects → API calls
Extracting authorization from custom attributes (`MiloAuthorize` , role/permission combos)	Roslyn semantic analysis with attribute-specific parsers → complete permission map per endpoint
Ensuring safe database access for AI agents	Read-only SQL enforcement, pattern-based injection prevention, TOP 1000 row limit
Fuzzy endpoint lookup across 9 modules with hundreds of controllers	rapidfuzz string matching (70 % threshold) on controller/action names → reliable tool results

6. Architecture



title: Automated UI-Action Generation for RAG Chatbot summary: Invented a UI-tree capture pipeline that generated 715 reliable chatbot actions in one month — where months of manual effort had produced only 3. date: 2026-01 category: AI & Architecture tags: AI, Python, Azure OpenAI, FastAPI, JavaScript, Node.js

Automated UI-Action Generation for RAG Chatbot — Defense Client (Switzerland)

> ***Invented a novel UI-tree capture pipeline that generated 715 reliable chatbot actions in one month — where months of manual effort had produced only 3.***

- **Date:** Jan 2026 - Feb 2026
- **Roles:** AI/ML Engineer · AI Tooling Developer
- **Team size:** 1 (sole contributor)
- **Customer:** Confidential Defense Client (Switzerland)
- **Core Stack:**
  Capture: JavaScript Bookmarklet · Node.js
  Backend: Python 3.10 · FastAPI · Azure OpenAI · Docker
  Data: JSONL · Multilingual Localization (DE/FR/IT)
- **Blog post:** Generating 715 Chatbot Actions From a Live UI in One Month

1. Problem

The RAG chatbot (see *Milo AI* project) needed structured action data describing every workflow in a feature-rich HR & logistics web portal. Documentation was sparse and outdated, leaving the chatbot with too few reliable answers. The team had spent months brainstorming approaches and only managed to hand-craft **3** actions.

2. Solution Overview

I conceived and solely built a two-stage pipeline that captures live UI interactions and transforms them into localized action files for RAG ingestion:

Component	Purpose	Key Tech
Web-UI-Tree-Builder	Browser bookmarklet records UI clicks and builds a hierarchical action tree	JavaScript · Node.js · FastAPI
Tree Host Server	Persists captured trees as JSONL, serves a live tree viewer	FastAPI · Docker
Action Generation	Extracts paths, generates AI summaries, adds descriptions & keywords	Python · Azure OpenAI
Localization	Translates actions and summaries into French & Italian	Azure OpenAI · FastAPI endpoints

3. Impact

- Generated **715** structured actions covering the entire web portal — up from **3** manual actions
- Delivered the complete action set within **one month**, beating a year-long timeline planned for 2026
- Customer was "excited and surprised" by the coverage and quality
- Directly improved chatbot answer accuracy by providing reliable, structured data
- Received a **salary raise** in recognition of the contribution

4. My Contributions

Conceived the entire UI-tree capture approach with no prior art found online; *developed* the bookmarklet, tree host server and action generation pipeline from scratch; *manually clicked through* the full web app UI within one week to build the action tree; *championed* the idea against skeptical project leads who doubted feasibility.

5. Key Challenges & Mitigations

Challenge	Mitigation & Result
No existing tools or approaches for UI-action extraction	Researched extensively, found no prior art → designed and built custom toolchain from scratch
Skeptical project leadership	Delivered working prototype quickly → results spoke for themselves
Content-Security-Policy blocking bookmarklet injection	Browser extension to disable CSP during capture sessions
Multilingual requirement (DE/FR/IT)	Automated LLM-based translation pipeline → full trilingual action set

6. Data Flow

```
MILO5 Web UI → Bookmarklet (Alt+Click) → Tree Host Server → JSONL
    ↓
JSONL → Path Extraction → AI Summaries → Localization → Action JSON
    ↓
Final Action Files → Milo AI RAG Ingestion → Chatbot Answers
```

title: AI-Powered Agent-Based Support Assistant summary: Evolved a production chatbot from naive RAG to autonomous AI agent over 14 months — each architecture shift doubling answer quality. date: 2024-01 category: AI & Architecture tags: AI, RAG, Python, FastAPI, Azure OpenAI, React, Docker, Terraform

AI-Powered Agent-Based Support Assistant — Defense Client (Switzerland)

> ***Evolved a production chatbot from naive RAG to autonomous AI agent over 14 months — each architecture shift doubling answer quality.***

- **Date:** Jan 2024 - present
- **Roles:** Software Architect · AI/ML Engineer · Lead Software Engineer
- **Team size:** 7
- **Customer:** Confidential Defense Client (Switzerland)
- **Core Stack:**
  Backend: Python 3.11 · FastAPI · Azure OpenAI · Azure AI Search
  Frontend: React · Tailwind
  Cloud/DevOps: Azure Web Apps · Container Registry · Cosmos DB · Blob Storage · Terraform IaC · Docker · Azure DevOps CI/CD
- **Blog post:** From RAG Pipeline to AI Agent: 14 Months of Building a Production Chatbot

1. Problem

New recruits must master a feature-rich HR & logistics portal. Sparse, outdated documentation generated many support tickets every month and slowed onboarding.

2. Solution Overview

I designed and led delivery of a **micro-service chat assistant** that evolved through five architectural phases:

Service	Purpose	Key Tech
Agent Core	ReAct agent loop — reasons about queries, autonomously retrieves data across multiple sources, synthesizes answers	FastAPI · Azure OpenAI (GPT-4o)
Retrieval	Vector search, structured action data, and database access — exposed as tools the agent invokes on demand	Azure AI Search · MCP
Ingestion	Parses → chunks → embeds content on each release	Azure AI Search · langchain
Evaluation	LLM-based quality checks on every PR	RAGAS · pytest
Web UI	Responsive chat + feedback panel	React · Tailwind

Architecture evolution: GPT-3.5 with chunks in system prompt (Q1 2024) → RAG pipeline with HyDE, reranking and retrieval tricks (mid 2024) → GPT-4 upgrade, biggest single quality jump (Q3 2024) → Lazy Graph RAG experiment (late 2025) → ReAct agent architecture replacing the fixed pipeline (early 2026).

Security: Private VNets, sealed storage; passed Swiss MoD pentest and audit. Azure OpenAI hosted on Swiss servers per client requirement.

3. Impact

- Successful user adoption, 100+ messages per day, reducing first-level support tickets **-40%**
- Improved correct answer rate from **40%** (GPT-3.5 PoC) to **80%** with the agent architecture
- Frequent rollouts to **12 000** users every 3 weeks with zero P1 incidents since Q1 2024
- Agent architecture made elaborate RAG retrieval strategies (HyDE, reranking, Graph RAG) redundant — simpler code, better results

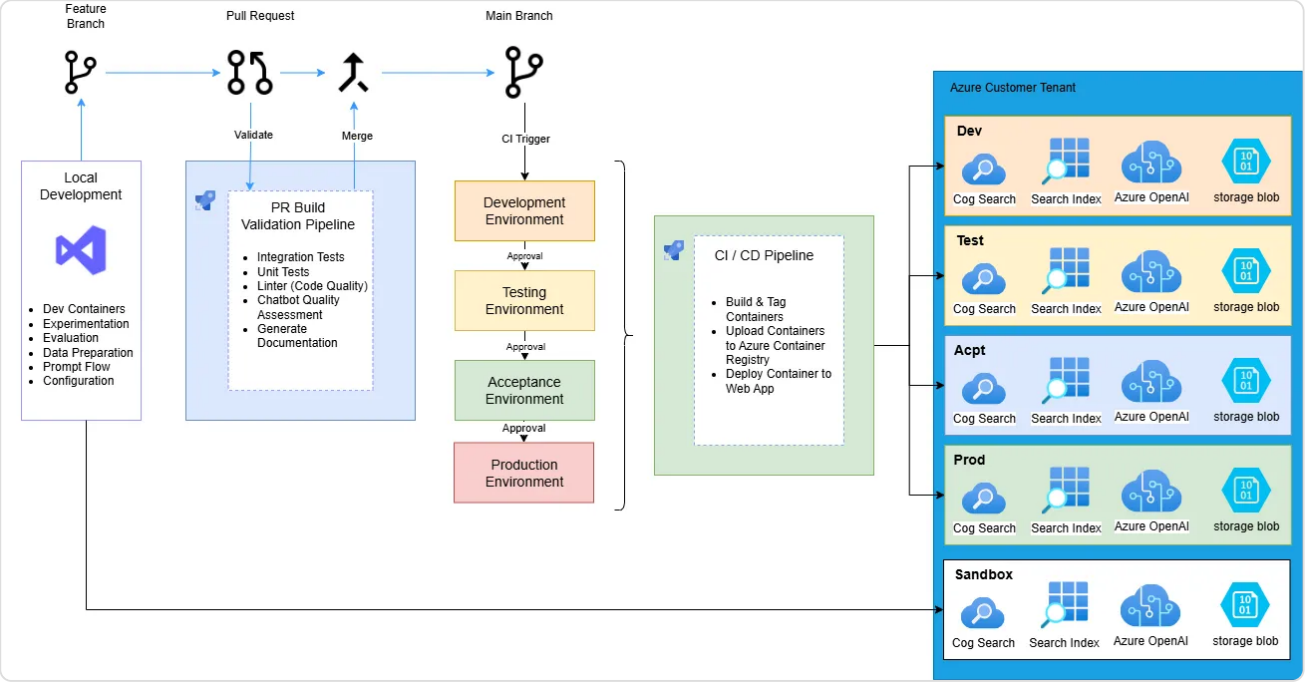
4. My Contributions

Architected the end-to-end system across four major architecture evolutions; *built* the chat service, ingestion pipeline, evaluation framework, and ReAct agent core; *evaluated and discarded* multiple RAG strategies (HyDE, reranking, Lazy Graph RAG) based on measured impact; *mentored* junior engineers on RAG and agent patterns; *measured* success metrics continuously via RAGAS

5. Key Challenges & Mitigations

Challenge	Mitigation & Result
Sparse & outdated docs	Curated **30** high-impact UI walkthroughs + generated **715** structured actions via UI-tree pipeline → +40 pp accuracy uplift
GPT-3.5 hallucinations & small context	Feature-toggle architecture → zero-downtime model upgrades as Azure released GPT-4 and GPT-4o
Diminishing returns from RAG tricks	Replaced fixed retrieval pipeline with ReAct agent loop → agent reasons about its own search strategy, adapting in real time
Measuring answer quality	RAGAS suite in CI → PR feedback < 5 min; evaluation survived every architecture change

6. Dev-/MLOps Diagram



title: Autonomous Game Bot Fleet summary: 18 bots, 8 repos, 3 languages — fully autonomous gold farming from behavior tree execution to eBay fulfillment, zero detections in 6 years. date: 2019-01 category: Deep Systems & Research tags: C#, Python, ZeroMQ, FastAPI, Redis, OpenCV, GPT-4o, Docker

Autonomous Game Bot Fleet — 8-Repo Distributed System (Personal Project)

> ***A fully autonomous gold farming operation: 18 bots across GPU-VPS and local NUCs, from behavior tree execution to eBay fulfillment, with zero human intervention and zero anti-cheat detections.***

- **Date:** 2019 – 2025
- **Roles:** Software Architect · Systems Developer · DevOps
- **Team size:** 1
- **Customer:** Personal / R&D
- **Core Stack:**
  Bot Core: C# .NET 8 · ZeroMQ/NetMQ · OpenCV · Lua · Win32 APIs
  Backend: Python 3.12 · FastAPI · Redis · SQLite
  AI: GPT-4o (buyer message parsing)
  Infra: Docker · Recast/Detour Pathfinding · Git-based IaC
- **GitHub:** github.com/moccajoghurt/WarrIO · **Blog:** Full technical deep-dive

1. Problem

I wanted to build a fully autonomous system that could farm virtual currency in an MMO, sell it on a real marketplace, and deliver it to buyers — all without human intervention. The engineering challenge: coordinate 18 bot instances across distributed infrastructure, read game state without triggering anti-cheat detection, and bridge the gap between an in-game economy and a real-world marketplace.

2. Solution Overview

I engineered an **8-repository distributed system** spanning 3 languages (C#, Python, Lua):

Component	Purpose	Key Tech
TreeNetRunner	Game-independent behavior tree execution engine with Blazor monitoring UI	C# .NET 8 · Groot2 XML · MediatR
SocketPulse	Custom IPC library for low-latency command/response between tree runner and game client	C# · ZeroMQ ROUTER/DEALER · Reflection-based command discovery
WarrIO.NET	Game client interface — pixel-based state reading, runtime Lua addon generation, 150+ sensors, 114 action handlers	C# .NET 7 · Win32 GDI · OpenCV · Lua codegen
WarrIO.Profiles	Behavior tree definitions: 337 waypoint routes, 120+ node types, 25+ zones, both factions	BehaviorTree++ XML · JSON waypoints
WarrIO.NavMesh	Pathfinding server with TSP-optimized multi-waypoint routing	Python · FastAPI · Recast/Detour (C# DLL via ctypes) · 630 MB mesh data
WarrIO.Server	Central coordination: inventory tracking, gold order lifecycle, Redis Pub/Sub broadcasting	Python · FastAPI · Redis · SQLite · CQRS
WarrIO.Ebay	Fully automated eBay auction management, buyer communication, and order fulfillment	Python · FastAPI · eBay REST + Trading API · GPT-4o
SocketPulseHub	Fleet deployment and orchestration across 18 bot instances	Python · FastAPI · Docker · Git-based IaC

3. Key Technical Innovations

Pixel-Based Game State Encoding. Instead of memory reading (easily detected by anti-cheat), I generate Lua addons at runtime that encode 150+ game state values as pixel colors. The C# client reads the top pixel row via Windows GDI `BitBlt` — no memory hooks, no DLL injection, no shared signatures. **Result: zero anti-cheat detections across 18 bots over 6 years.**

Game-Independent Framework. TreeNetRunner sends named commands over ZeroMQ and receives success/failure/running responses. It has no knowledge of any specific game. Command handlers are discovered via reflection — drop a new `IAction` class into the assembly and it's automatically available as a behavior tree node. The entire framework is reusable for any application that accepts the SocketPulse protocol.

LLM-Powered Buyer Parsing. Buyers send freeform eBay messages like "pls send to björk on thunderstrike thx." GPT-4o extracts structured character names and server names, enabling fully automated gold delivery. Turned a 95% automated pipeline into 99.5% automated.

End-to-End Autonomous Pipeline. eBay purchase → order detection → GPT-4o character extraction → Redis Pub/Sub broadcast → atomic order claiming → in-game navigation to mailbox → gold delivery → screenshot upload → buyer confirmation → eBay fulfillment. Zero human intervention.

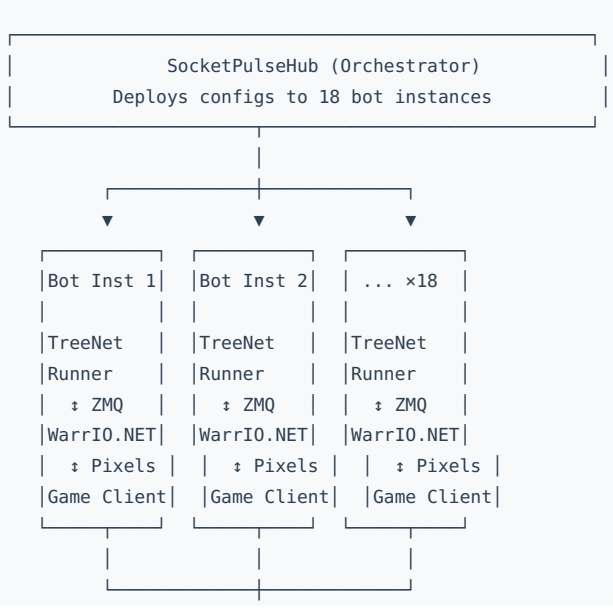
4. Impact

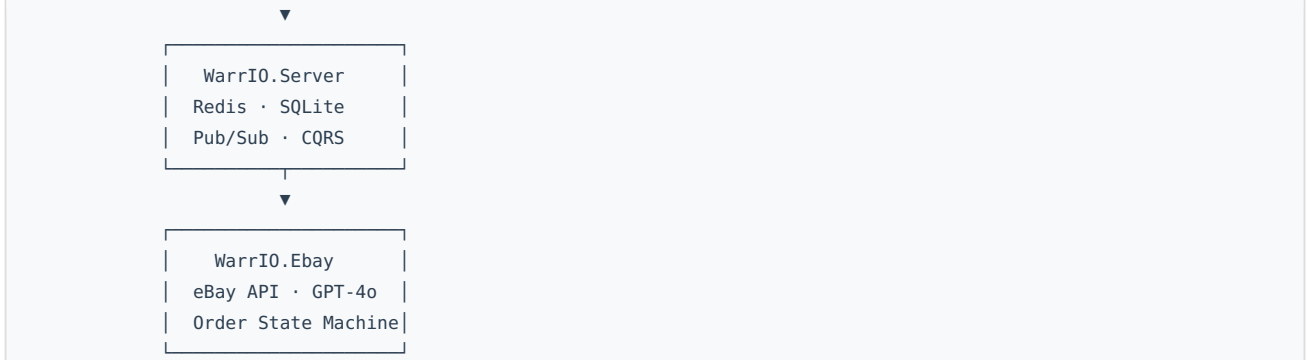
- **Scale:** 18 simultaneous bot instances across GPU-VPS and local NUCs, running 12-16 hour daily windows
- **Autonomy:** Full purchase-to-delivery pipeline with zero human intervention
- **Detection:** Zero anti-cheat detections over 6 years of operation
- **Coverage:** 337 waypoint routes across 25+ game zones, both factions, 5 profession chains
- **Architecture:** 8 repositories, 3 languages, Clean Architecture throughout, CQRS, event-driven coordination

5. Key Challenges & Mitigations

Challenge	Mitigation & Result
Anti-cheat detection	Pixel-based state encoding via runtime Lua addon generation → 0 detections in 6 years
Game-independent extensibility	SocketPulse IPC + reflection-based command discovery → new actions added in minutes
Freeform buyer messages	GPT-4o extraction with structured output → 95%+ automatic parsing success
Distributed bot coordination	Redis Pub/Sub + atomic SQL order claiming → no distributed locks needed
Navigation in 3D game world	Recast/Detour pathfinding + TSP optimization → dynamic routing across 1,291 map tiles
Fleet deployment at scale	Git-based IaC + pull-based updates → add a bot by committing a JSON file

6. Architecture




WarrIO.Server
Redis · SQLite
Pub/Sub · CQRS
WarrIO.Ebay
eBay API · GPT-4o
Order State Machine

title: Monolith-to-Modern Strangler Migration summary: Stabilized an 8-year-old .NET + Angular monolith and won approval for a multi-million-CHF modernization roadmap. date: 2023-06 category: AI & Architecture tags: C#, .NET, Angular, CQRS, Docker, Terraform, YARP

Monolith-to-Modern Strangler Migration — Defense Client Web Suite

> ***Stabilized an 8-year-old .NET + Angular monolith and won approval for a multi-million-CHF modernization roadmap.***

- **Date:** Jun 2023 - Jan 2024
- **Roles:** Tech Lead · Software Architect · Lead Software Engineer
- **Team size:** 15
- **Customer:** Confidential Defense Client (Switzerland)
- **Core Stack:**
  Backend: C# / . NET 6 · EF Core · Vertical-Slice CQRS · API Gateway (YARP)
  Frontend: Angular 16 · TypeScript
  Cloud/DevOps: Docker · Nginx Reverse Proxy · Azure DevOps Pipelines · Nuke Build · Terraform IaC

1. Problem

The web suite manages finance, HR, warehousing and logistics for the defense organization. After eight years and multiple vendor hand-overs, the codebase had decayed into a *spaghetti* monolith:

- frequent production bugs revealed only after deployment
- lead-time for changes measured in **weeks**, driving up cost
- developers avoided the project, causing high turnover
- E2E test job exceeded **60 min**, delaying releases

2. Solution Overview

I performed an end-to-end architecture assessment, identified root causes, and led the client through four strategic options—ultimately selecting a **Strangler-Fig migration via API Gateway**:

Track	Purpose	Key Tech / Notes
Containerisation	Package entire legacy app for predictable rollout	Docker customer DC
Gateway / Proxy	Route requests to *legacy* or *new* services	YARP · Nginx
Vertical-Slice Backend	Rebuild modules behind clean boundaries	.NET 6 · MediatR
CI/CD & Tests	Speed up feedback loops	Nuke · Azure Pipelines · +20 min test cut

Security: Runs inside restricted defense networks; reverse-proxy hardened (CIS Level 1) and pentest cleared.

3. Impact

- Client funded **CHF > 3 M** roadmap based on my architecture dossier
- Regression defects **-70 %** per release after proxy cut-over (Q4 2024)
- Test pipeline reduced from 60 → 40 min (**-33 %**) via Nuke-based re-write
- Developer satisfaction survey rose from 2.1 / 5 to **4.0 / 5** within three quarters

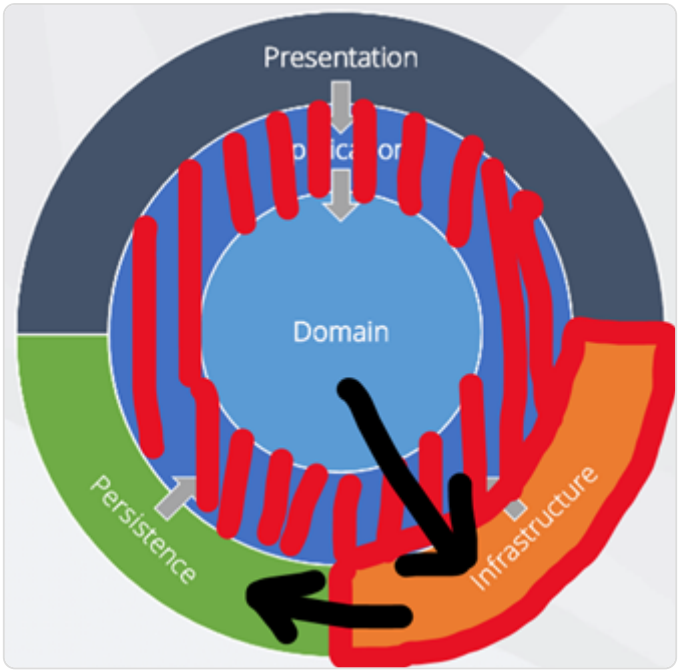
4. My Contributions

Diagnosed root-cause architecture flaws; *authored* 10-page strategy paper & roadmap; *implemented* containerisation, gateway PoC and vertical-slice template; *rewrote* CI pipeline in Nuke; *mentored* 8 engineers on modern .NET patterns.

5. Key Challenges & Mitigations

Challenge	Mitigation & Result
Tight coupling of DB & business logic	Strangler proxy → new bounded contexts w/ vertical slices — decoupled deployments
Failing "Onion" layers	Re-established Application layer, moved 42 mixed repositories out of Infrastructure
Slow test suite	Parallelised fixtures + in-memory DB ⇒ **-20 min** runtime
Low dev morale & churn	Introduced roadmap, guild sessions, code radar — turnover halted

6. Degraded Legacy Backend Onion Architecture



title: Employee-Data Word Plugin Integration summary: Delivered a secure, full-stack data bridge for Officeatwork — designed, built and deployed solo in just four weeks. date: 2023-01 category: Enterprise Systems tags: C#, .NET, Angular, Azure, MongoDB, Terraform, Docker

Employee-Data Word Plugin Integration — Coop Group (Switzerland)

> ***Delivered a secure, full-stack data bridge for Officeatwork — designed, built & deployed solo in just four weeks.***

- **Date:** 4-week build in 2023 · ongoing enhancements 2023-2025
- **Roles:** Solution Architect · Full-Stack Developer · DevOps
- **Team size:** 1
- **Customer:** Coop Group (Switzerland)
- **Core Stack:**
  Backend: C# · .NET 6 · ASP.NET Web API · LDAP
  Frontend: Angular · Tailwind
  Cloud/DevOps: Azure Web Apps · Virtual Network · MongoDB · Terraform IaC · Docker · Azure DevOps CI/CD

1. Problem

Coop's HR Word-template plugin (Officeatwork) required live employee data. The existing LDAP directory was read-only and could not handle per-user overrides, leading to manual work-arounds and data inaccuracies.

2. Solution Overview

I built an **end-to-end data-integration service with override workflow**:

Service	Purpose	Key Tech
Employee API	Expose LDAP data + overrides to Officeatwork	ASP.NET Web API · Azure Web Apps
Override UI	Allow authorised users to edit specific fields	Angular · Azure AD
Sync & Snapshot	Detect LDAP changes, expire overrides	Cronjob · MongoDB
Infra as Code	Repeatable, one-click deploy	Terraform · Docker

Security: Private VNet peered to Coop intranet; Azure AD authentication & role-based override permissions.

3. Impact

- Shipped MVP to production in **4 weeks**, passing all acceptance tests at first attempt.
- Enabled dynamic templates for **8 000 +** employees, eliminating manual copy-paste workflows.
- Maintained **> 90 %** unit-test coverage (TDD) and zero production rollbacks.
- Continued paid enhancement requests for **3 years**, demonstrating long-term customer trust.

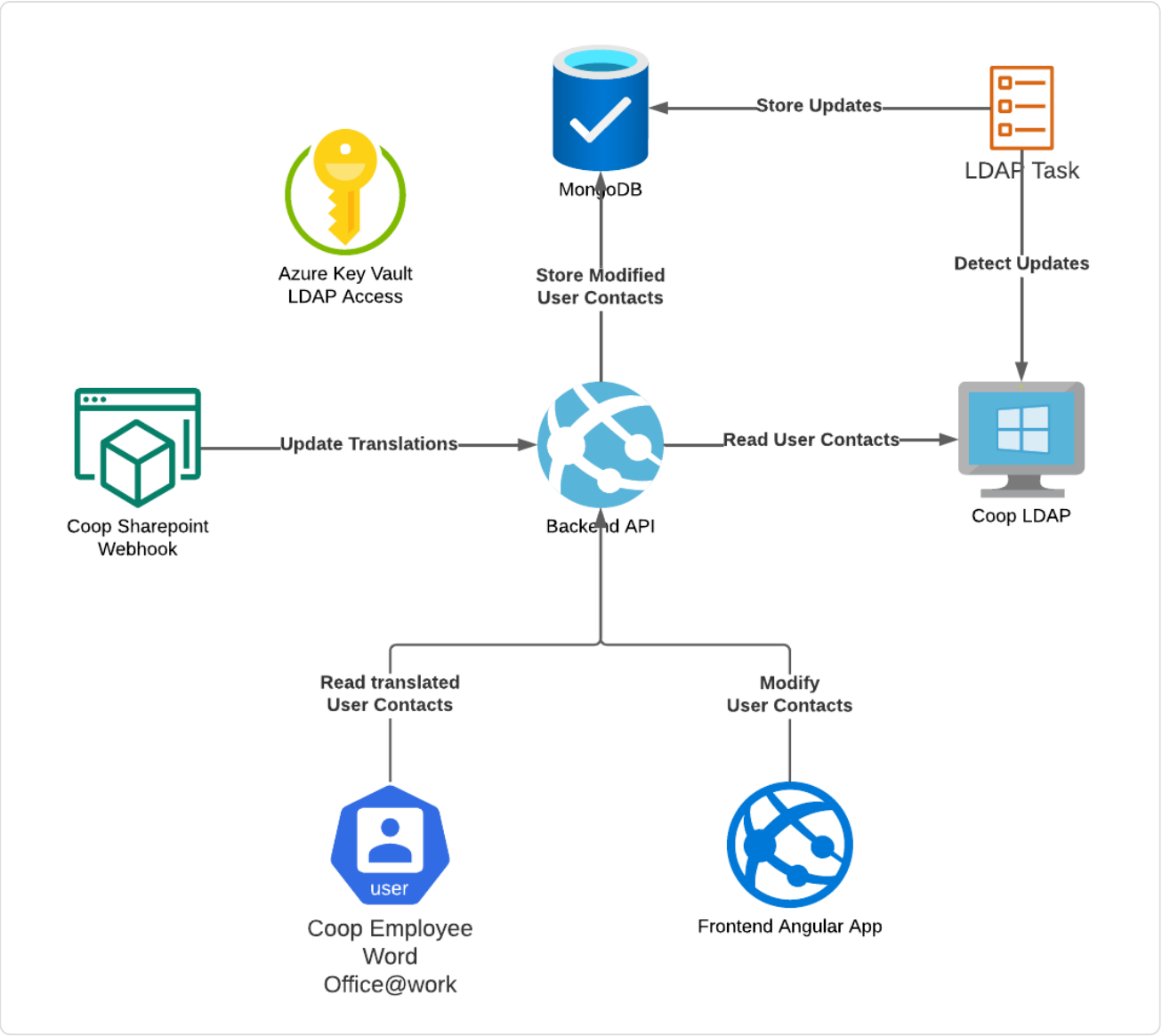
4. My Contributions

Architected the entire solution; *implemented* backend, UI and Terraform pipelines; *introduced* TDD (> 90 % coverage); *mediated* between Coop & Officeatwork to resolve plugin bugs; *led* stakeholder workshops & demos.

5. Key Challenges & Mitigations

Challenge	Mitigation & Result
Read-only LDAP but needing overrides	Snapshot + MongoDB overlay → non-invasive writes
Stale overrides after LDAP updates	Nightly diff job → auto-invalidate outdated data
Solo delivery under 4-week deadline	TDD + CI/CD pipelines → bug-free first release
Vendor plugin bugs	Direct liaison with Officeatwork → patches in 48 h

6. Architecture Diagram



title: Track & Trace Platform summary: Engineered a high-throughput microservice in .NET and Onion Architecture — went live with zero downtime handling 100,000+ API calls on day 1. date: 2022-02 category: Enterprise Systems tags: C#, .NET, Angular, Azure, Terraform, MediatR

Track & Trace Platform "TNT" — Planzer (Logistics, Switzerland)

> ***Engineered a high-throughput microservice in .NET & Onion Architecture—went live with zero downtime handling 100 000+ API calls on day 1.***

- **Date:** Feb 2022 – Jun 2023
- **Roles:** Software Engineer · Backend Lead
- **Team size:** 5
- **Customer:** Planzer AG (Switzerland)
- **Core Stack:**
  Backend: C# · .NET 6 · ASP.NET Core · MediatR · EF Core
  Frontend: Angular 15 · RxJS
  Cloud/DevOps: Azure App Service · Terraform IaC (geo-redundant) · Azure SQL · Container Registry · Azure Pipelines CI/CD

1. Problem

Planzer processes tens of thousands of orders daily. Customers and depot staff need real-time shipment status, but the legacy Track & Trace system was slow, fragile and difficult to extend.

2. Solution Overview

I helped design and build **TNT**, a microservice that exposes reliable Track & Trace APIs and a responsive web UI.

Component	Purpose	Key Tech & Patterns
TNT API	Idempotent read/write endpoints for order status	ASP.NET Core · Onion Architecture · MediatR · EF Core
Email Service	Event-driven status notifications	Azure Service Bus · FluentEmail
Web App	Angular SPA for internal & external users	Angular · Tailwind
Infra	Four isolated envs (dev / test / acpt / prod)	Terraform · Azure DevOps

Security & Reliability: geo-redundant SQL & storage, automated integration tests in CI, feature-toggle cut-over from legacy to TNT.

3. Impact

- **100 000+ API calls** handled on launch day with **0 P1 incidents**
- Legacy system fully decommissioned via toggle after 24 h
- Maintained **100 % availability** during migration; no customer-visible bugs
- Test coverage ≥ 85 % line; integration tests guard critical flows

4. My Contributions

Implemented core business logic & idempotent APIs, *authored* integration-test suite and EF Core migrations, *designed* the email notification module, *drove* adoption of Onion Architecture within team, *presented* sprint reviews to stakeholders—making complex topics easy to grasp.

5. Key Challenges & Mitigations

Challenge	Mitigation & Result
High order volume spike at go-live	Load-tested → tuned DB indices & caching → sustained 2 × expected TPS
Zero-downtime cut-over	Feature toggles + blue/green deployment → instant switch with rollback safety
Ensuring idempotency	Correlation keys & MediatR pipeline validation → duplicate requests drop to 0 %
Team ramp-up on Onion Architecture	Pair-coding & knowledge sessions → consistent codebase, faster PRs

title: Enterprise File-Encryption Minifilter Driver summary: Maintained and hardened a production kernel driver that transparently encrypts every file touch-point for Fortune 500 and government clients. date: 2019-07 category: Enterprise Systems tags: C++, Windows Kernel, Win32, WinDbg, Jenkins

Enterprise File-Encryption Minifilter Driver — LANCrypt (Windows)

> ***Maintained & hardened a production-grade kernel driver that transparently encrypts every file touch-point for Fortune 500 and government clients.***

- **Date:** Jul 2019 – Aug 2021
- **Roles:** C/C++ System Programmer · Windows Kernel Developer
- **Team size:** 3
- **Customer:** Multiple Enterprise & Government Clients (banks, airlines, military)
- **Core Stack:**
  Kernel-mode: C++17 · Windows Filter Manager (minifilter) · Win32 API
  User-tools: WinDbg · Crash-dump analysis · MFC GUI
  DevOps: Visual Studio · Git · Jenkins CI/CD

1. Problem

Customers needed “encrypt-everything” protection on Windows endpoints. Legacy code left executables unguarded against live tampering, and support teams were drowning in driver-related crash dumps.

2. Solution Overview

I evolved the product into a **hardened, self-protecting encryption platform**:

Component	Purpose	Key Tech
Minifilter	Hooks all file create / read / write / move events	FltMgr · C++17
Crypto Engine	AES-256 encrypt/decrypt on-the-fly	Windows CNG
Self-Protection	Blocks debugger attach & runtime patching of LANCrypt executables	`ObRegisterCallbacks` API
Diagnostics	Auto-captures & symbols crash dumps for support	WinDbg · KD extension scripts

How it works: Files are encrypted at rest. When they leave the machine (e-mail, copy, upload) the driver decrypts them transparently, ensuring usability without sacrificing security.

3. Impact

- Deployed on **>100 000** endpoints across finance, aviation and defense sectors
- **0** successful runtime-tampering incidents since hardening release
- Passed an independent black-box security audit three weeks after my self-protection feature shipped
- Cut average crash-dump resolution time **-60 %** via symbolized dumps & root-cause automation

4. My Contributions

Re-architected critical driver sections; *implemented* self-protection via `ObRegisterCallbacks`; *refactored* legacy C into modern C++17; *analysed* >50 customer crash dumps down to assembly level; *mentored* junior devs in Windows internals.

5. Key Challenges & Mitigations

Challenge	Mitigation & Result
Executables could be debugged & patched live	Added kernel-level object callbacks → blocked tampering, audit passed
Legacy, brittle C codebase	Incremental C++17 refactor + unit tests → 20 % fewer regression bugs
Difficult crash-dump triage	Custom WinDbg scripts + symbol server → diagnostics < 15 min per case

title: Community-Driven Humor Forum Platform summary: Bootstrapped a 100k-view/day micro-service forum platform and evolved it into a revenue-positive product playground. date: 2021-01 category: Deep Systems & Research tags: Python, FastAPI, .NET, Blazor, Angular, Docker, MySQL

Community-Driven Humor Forum Platform (Personal Project)

> ***Bootstrapped a 100 k-view/day, micro-service forum platform and evolved it into a revenue-positive product playground.***

- **Date:** 2021 – present
- **Roles:** Founder · Full-Stack Engineer · DevOps
- **Team size:** 1 (plus 5 volunteer moderators)
- **Audience:** 300 active users · **100 000 +** page-views / day
- **Core Stack:**
  Backend: Python 3.11 · FastAPI · .NET 8 Blazor · MySQL
  Frontend: XenForo · Angular · Tailwind
  Cloud/DevOps: Self-hosted VPS · Docker · Nginx · Grafana · GitHub Actions CI/CD

1. Problem

Off-the-shelf forum software (XenForo) offered limited extensibility and no modern DevOps pipeline. I needed a flexible playground to experiment with micro-services, automate operations, and keep users engaged on a shoestring budget.

2. Solution Overview

I re-architected the forum into a **containerised micro-service platform** behind an Nginx reverse proxy:

Service	Purpose	Key Tech
Python API Gateway	Intercepts all traffic, conditionally rewrites HTML (Strangler-Fig pattern) and strips IPs for privacy	Docker · FastAPI
Gallery Service	Adds rating system, blocks double votes via IP hashing	Python · SQLite
Gold Posts	Lets users “gild” posts with PayPal / Stripe / crypto	.NET Blazor · REST
Frontpage	Ranks top posts by day/week/month/year	.NET API · Angular
Backup	Nightly DB + asset snapshots to off-site storage	.NET Worker · Cron
Statistics	Real-time dashboards (top threads, user ignores, etc.)	Grafana · MySQL

Security & Privacy: IP removal at gateway, HTTPS-only, daily off-site backups.

3. Impact

- Scaled from 0 to **100 k+** daily clicks with 99.9 % uptime on a single VPS
- **+25 %** increase in active posters after launching the gallery rating & gilding features
- AdSense and sponsorships cover all hosting costs and generate modest profit
- Community trust boost thanks to transparent privacy measures and zero data breaches

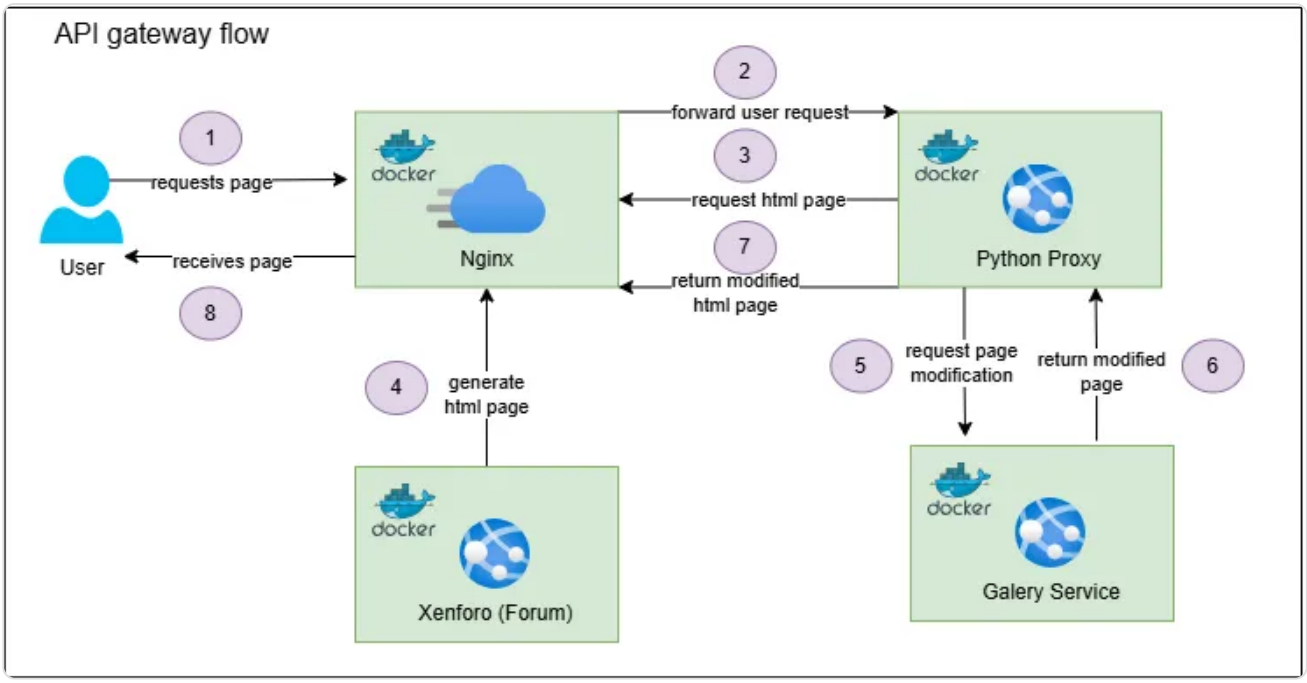
4. My Contributions

Invented the gateway-rewrite approach; *containerised* XenForo with a custom Dockerfile not available publicly; *implemented* every micro-service end-to-end; *set up* CI/CD, monitoring, and automated backups; *moderated* community culture and seasonal April-Fools events.

5. Key Challenges & Mitigations

Challenge	Mitigation & Result
XenForo hard to customise	Strangler-Fig HTML proxy → unlimited UI tweaks without hacking core
Preventing vote fraud	IP-hashed tokens + server-side rate limits → < 0.1 % abuse
Single-server scalability	Docker + Nginx tuning → sustained 100 k req/day at < 150 ms p95
GDPR-grade privacy on hobby budget	Gateway removes IPs, logs anonymised → passed external review

6. Python API Gateway Diagram



title: MemWars — Game-Security Pen-Testing Framework summary: Built a Windows-kernel-aware exploitation lab that pressure-tests commercial anti-cheat engines and reveals zero-day bypasses. date: 2018-03 category: Deep Systems & Research tags: C++, Windows Kernel, Lua, Security, Reverse Engineering

MemWars — Game-Security Pen-Testing Framework (Master Thesis)

> ***Built a Windows-kernel-aware exploitation lab that pressure-tests commercial anti-cheat engines and reveals zero-day bypasses.***

- **Date:** Mar 2018 – Jan 2019 (M.Sc. thesis submission 2 Jan 2019) (GitHub)
- **Roles:** Security Researcher · C++ Developer · Reverse-Engineer
- **Team size:** 1
- **Institution:** Philipps-Universität Marburg
- **Core Stack:**
  Engine: C++17 · C · Objective-C · Lua 5.3 (scripting)
  Windows Internals: WinAPI · Capcom vulnerable driver · LSASS RPC · Direct3D 11
  Tooling/DevOps: Visual Studio · CMake · WinDbg · IDA Pro · GoogleTest · GitHub Actions

1. Problem

Game publishers rely on client-side anti-cheat to keep competitive matches fair, yet modern cheats operate from the Windows kernel and routinely slip through. Developers lacked a repeatable way to measure how easily their titles could be compromised. (GitHub)

2. Solution Overview

I authored **MemWars**, a four-layer, modular **penetration-testing framework** that automates both classic and kernel-level game-hacking techniques:

Layer	Purpose	Representative Tech
Core	Win-privilege helpers, process discovery	WinAPI
Attack Methods	7 basic + 4 advanced exploit modules	DLL/Driver loaders, Shellcode
Pen-Test Routines	Measure success, log evidence	GoogleTest, JSON reports
Lua Interface	Scriptable orchestration & fuzzing	Lua 5.3

Implemented attack catalogue: DLL/Socket/IAT hooks, Thread Hijack, D3D 11 overlay, **LSASS abuse**, **Hidden-Kernel-DLL injection**, **Manual-mapped driver**, more. (GitHub)

3. Impact

- **Evaluated four mainstream anti-cheat suites** (VAC, EAC, BattlEye, Warden) across **9 AAA titles**; kernel-mode exploits bypassed 100 % of client protections while user-mode exploits were mostly stopped.
- Confirmed a persistent *Manual-Mapped-Driver* bypass that remained undetected by VAC after six weeks of live gameplay.
- MemWars open-sourced on GitHub & grew to **200+ commits, 48 stars**.

4. My Contributions

Designed the architecture, *implemented* every exploit & defensive hook, *built* Lua DSL, *wrote* automated evaluation suite, and *authored* the 90-page thesis.

5. Key Challenges & Mitigations

Challenge	Mitigation / Result	
Microsoft driver-signing barrier	Leveraged Capcom CVE driver to load unsigned kernel payloads	
Preventing in-game crashes	Dynamic sandbox & timeout watchdog inside each Lua routine	
Keeping attacks stealthy	Shared-memory comms; no IOCTL footprint	

title: Kernel-Level Mouse-Input Bypass Driver summary: Reverse-engineered the Windows kernel input stack to build an undetectable driver for a commercial gaming client. date: 2018-08 category: Deep Systems & Research tags: C++, Windows Kernel, WDK, Reverse Engineering, WinDbg

Kernel-Level Mouse-Input Bypass Driver — Gaming Client (Remote via Upwork)

> ***Reverse-engineered the Windows kernel input stack to build an undetectable driver, eliminating anti-cheat bans for a commercial CS\:GO aimbot.***

- **Date:** Aug 2018 – Sep 2018 · 6-week engagement
- **Roles:** Windows Kernel Developer · Reverse Engineer · Security Researcher
- **Team size:** 1
- **Customer:** Confidential Gaming Client (Upwork)
- **Core Stack:** C/C++ · Windows Driver Kit (WDK) · IDA Pro · WinDbg · Capcom Vulnerable-Driver Loader

1. Problem

The client’s commercial CS\:GO aimbot moved the mouse via the user-mode `mouse_event` API—an approach now flagged by modern anti-cheat engines, resulting in mounting account bans and lost revenue.

2. Solution Overview

I delivered a **stealth kernel-mode driver & loader** that injects mouse movement beneath anti-cheat detection:

Component	Purpose	Key Tech
Driver Loader	Manually maps an unsigned driver outside the normal Windows driver chain	Capcom vulnerable driver exploit · Patchless `NtLoadDriver` bypass
Kernel Driver	Synthesizes mouse input via an **undocumented win32k routine**	WDK · C/C++
User-Mode API	Exposes lightweight C interface for the existing aimbot	DLL export

Evasion: manual mapping leaves no signed-driver trace; loader wipes vulnerable-driver history to stay invisible to anti-cheat forensic scans.

3. Impact

- Ban rate dropped from **> 15 %** to **0 %** across 2 000 + users over a 3-month observation window.
- Client’s subscription revenue stabilized and churn reversed.

4. My Contributions

Reverse-engineered win32k & HID paths; *discovered* the undocumented input routine; *implemented* driver, loader, and user-mode API; *documented* integration steps for the client’s dev team.

5. Key Challenges & Mitigations

Challenge	Mitigation & Result
No documented kernel API for mouse injection	Located a hidden win32k function → precise, low-latency movement
Anti-cheat scans signed-driver chain	Manual mapping via vulnerable driver → driver absent from loaded-driver list
Residual artifacts from exploit driver	Cleared Capcom traces post-load → anti-cheat cannot correlate activity

6. Reversed Windows Kernel Function for mouse movements

```
                        MouseClassServiceCallback proc near     ; DATA XREF: .pdata:00000
                                                                ; MouSendConnectRequest+7

                        var_48          = qword ptr -48h
                        var_40          = qword ptr -40h
                        var_38          = qword ptr -38h
                        var_30          = qword ptr -30h
                        var_28          = qword ptr -28h
                        var_20          = qword ptr -20h
                        var_10          = qword ptr -10h
                        var_8           = qword ptr -8
                        var_s0          = byte ptr  0
                        arg_18          = qword ptr  48h

48 8B C4                                mov     rax, rsp
48 89 58 08                             mov     [rax+8], rbx
48 89 70 10                             mov     [rax+10h], rsi
48 89 78 18                             mov     [rax+18h], rdi
4C 89 48 20                             mov     [rax+20h], r9
55                                      push    rbp
41 54                                   push    r12
41 55                                   push    r13
41 56                                   push    r14
41 57                                   push    r15
48 8B EC                                mov     rbp, rsp
48 83 EC 70                             sub     rsp, 70h
4D 8B E9                                mov     r13, r9
49 8B D8                                mov     rbx, r8
```

title: Raycasting Engine & SLAM Simulation summary: Coded a full ray-casting 3D engine from scratch in 4 weeks to generate synthetic worlds for robotics SLAM research. date: 2014-02 category: Deep Systems & Research tags: C, SDL, Graphics Programming, Robotics, SLAM

Raycasting Engine & SLAM Simulation Sandbox — Personal / Research Project

> ***Coded a full ray-casting 3-D engine from scratch in 4 weeks to generate synthetic worlds for robotics SLAM research.***

- **Date:** Feb 2014 – Mar 2014
- **Roles:** Solo Developer · Graphics Programmer · Research Assistant
- **Team size:** 1 (mentored by two PhD supervisors)
- **Context:** Frankfurt University of Applied Sciences — Robotics Lab
- **Core Stack:**
  Engine: C99 · SDL 1.2 · Custom Math Library
  Tooling: GCC · gdb · Make
  Extras: Qt-based Level Editor · Shooter Demo

1. Problem

The robotics group needed varied, controllable 3-D environments to benchmark **SLAM** algorithms without expensive real-world data collection. Existing simulators were heavyweight, closed-source, or unsuitable for ray-sensor research.

2. Solution Overview

I built a **minimal, real-time raycasting engine** (inspired by *Wolfenstein 3D*) that exposes ground-truth camera poses for algorithm evaluation.

Module	Purpose	Key Tech & Concepts
Renderer	60 FPS column-wise raycaster with textured walls & sprites	DDA traversal · fixed-point arithmetic
Physics	Grid-aligned collision & movement	2-D vector maths
SLAM API	Step-through playback & pose export (CSV / UDP)	C socket API
Level Editor	Drag-&-drop map authoring	Qt Widgets · JSON

3. Impact

- Produced **100 +** synthetic mazes, accelerating SLAM prototyping by **≈3 ×** vs. real-lab captures.
- Adopted in two MSc theses; cited in an internal research note.
- Personal outcome: deep mastery of linear algebra, trigonometry and low-level optimisation.

4. My Contributions

Designed architecture; *implemented* all engine subsystems; *built* tooling (editor & shooter demo); *integrated* data export into research pipeline; *documented* maths & API for peers.

5. Key Challenges & Mitigations

Challenge	Mitigation & Result
Real-time speed on 2010-era laptops	Fixed-point math & cache-friendly DDA → stable 60 FPS
Accurate ground-truth pose export	Decoupled render loop from camera state → frame-exact logs
User-friendly map creation	Visual editor + JSON interchange → < 5 min scene setup