Anti-Cheat Privacy Monitoring via KDMapper & Kernel Callbacks

Project Deliverables – COMP6841 / Cyber Security Engineering

Author: Maxwell (zID: z5577783)

Date: 2025-07-25

1. Context & Initial Plan

Original Goal (from Week 1 plan):

Analyse privacy–security trade-offs in mainstream anti-cheat (AC) systems and propose concrete mitigations.

Planned Approach (My version):

- Reverse-engineer kernel/user-mode AC components
- Run dynamic traces to capture sensitive actions
- Apply threat modelling
- Deliver PoC least-privilege drivers & telemetry-hardening module

Planned Schedule Recap:

- **W1** Gather AC samples, build analysis environment
- **W2** Start reversing
- **W3** Static RE (Ghidra/IDA) to mark privacy-sensitive APIs
- **W4** Dynamic tracing & event DB design
- **W5** Legal gap analysis
- **W6** Flex week: script polishing, backlog catch-up
- **W7** Design & code least-privilege mitigator driver
- **W8** Compile final report, record demo video

Planned Deliverables:

1. Technical analysis report
2. Privacy-focused mitigation notes

3. PoC toolkit (demo)
4. Presentation or Video
5. Weekly progress blog

---

2. Final Outputs (Results)

After 40+ hours of work, I produced:

- **R1. Modified KDMapper**
  - Added **MDL / IndependentPages allocation modes** to stealth-map unsigned drivers.
  - Exposed allocation strategies via CLI flags and cleaned up PE fixups / cookie checks.
  - **Evidence:** E1 (patch diff), E2 (key path screenshots/notes)
- **R2. HelloWorld Kernel Monitor Driver (KMDF) + Privilege-Escalation & Evasion Chain**
  - Implemented process/thread/image-load callbacks to log AC modules.
  - Added selective hooks on `NtOpenProcess` / `NtReadVirtualMemory` with `PreviousMode` checks to cut noise.
  - **Inspired by Riot's anti-cheat engineer post** (https://revers.engineering/superseding-driver-altitude-checks-on-windows/), I built a **PageGuard-bypass, unsigned-driver privilege-escalation chain**:
    - Includes a **ValidSection integrity check bypass** (`je` → `jmp`) as part of the load chain (this is **not** a KDMapper feature).
    - Overcame kdmapper's traditional limitation where Windows handle validation blocks certain kernel APIs.
  - **Bypassed EAC/BE patch scanning** by avoiding common scan surfaces and performing **kernel lateral movement** (e.g., stealing privileged minifilter handles) to gain additional monitoring footholds.
  - Combined multiple monitoring techniques to comprehensively capture privacy-invasive actions.
  - **Evidence:** E3 (driver source/patches), E4 (DebugView/WinDbg logs)
- **R3. Privacy Event Logs (EAC & BE)**
  - ~5-minute captures each, quantifying memory-read counts, target processes/modules, sizes, and identifier types.
  - Public copies are redacted; originals are stored securely.
  - **Evidence:** E5 (redacted logs + legend)
- **R4. Demo Video**
  - Shows driver load, KDMapper patch usage, event capture, and quick analysis.
  - **Evidence:** E6 (video link)

- **R5. This Deliverables Document (+ Appendix)**
  - Replaces the earlier technical guide PDF while keeping full technical detail in the appendix.
  - **Evidence:** E7 (original PDF archived)

> **Proudest part:** Achieving PatchGuard-safe, minimally intrusive monitoring while still exposing concrete privacy events—demonstrating the feasibility of a least-privilege, detection-evasive inspection pipeline.

---

3. Plan vs Outcome (Delta Summary)

Item	Planned	Outcome	Reason/Comment
Static RE deep dive (W3)	Full API mapping in IDA/Ghidra	**Partially done**: focused on loader & memory access paths only	Time trade-off for dynamic logging pipeline
Event DB design (W4)	Dedicated DB schema	**Simplified**: Well Structed logs + Analysis	Faster iteration, enough for small dataset
Legal gap analysis (W5)	Formal comparative review	**Condensed** into ethical section & notes	Space/time limits in 2-page summary; still addressed
Mitigator driver (W7)	Full hardening module	**Pivoted** to monitoring driver + recommendations	Scope creep risk; mitigation ideas documented

---

4. What I Did (Process, Problems, Recovery)

Time Allocation (40h+ total):

- Driver design & implementation: ~20h
- KDMapper source study & patching: ~8h
- Data capture & log analysis: ~6h
- Documentation & video: ~6h

Key Steps:

1. **Safe baseline first**: implemented kernel callbacks before touching SSDT.
2. **Selective hooking**: Only `NtReadVirtualMemory` / `NtOpenProcess`, with mode checks to reduce noise.
3. **Mapper hardening**: MDL allocation & ValidSection bypass to survive AC integrity checks.
4. **Controlled testing**: VM isolation, KDNet debugging, frequent snapshots to recover from BSODs, Hyper-V / KVM.

Problems & Fixes:

- BSOD from invalid pointers → added pointer validation & _try / _except.
- KDMapper compatibility with PatchGuard → moved to MDL & independent pages.
- VS/WDK project compile error → fixed linker/INF settings.

---

5. How I Was Challenged (Reflection)

- **Technical stretch:** First hands-on with Windows MDL internals and PE manual mapping in kernel space.
- **Strategy to cope:** Modularise, snapshot often, instrument each stage, read MS docs and community code.
- **Ethical insight:** Techniques mirror malicious loaders, I restricted scope, redacted data, and drafted mitigations.
- **Personal growth:** Shifted from tool user to tool modifier; next time I'll automate parsing/visualisation earlier and explore a signed minifilter for FS events.

---

6. Strengths & Weaknesses

Strengths

- Working PoC with quantifiable outputs (event counts, time windows).
- Touched multiple layers: RE, kernel dev, loader patching, privacy analysis.
- Clear link to course objectives (engineering + ethics + analysis).

Weaknesses

- Short logging windows; may miss infrequent telemetry.
- No full-fledged minifilter/file audit.
- Manual triage for some logs (partial automation only).

7. Professional & Ethical Considerations

- Conducted all tests in isolated VMs; no online cheating / production impact.
- Highlighted misuse risks; recommended transparency APIs for AC vendors.
- Redacted sensitive info in shared logs; originals stored securely.
- Declared AI assistance (see §10).

8. Mapping to Course Objectives (Weeks 1–9)

- **Week 1 – Engineering Security**
  Designed a minimal-intrusion, PatchGuard-safe monitoring pipeline; used VM isolation + KDNet debugging to engineer failure domains and rollback paths.
- **Week 2 – Secrets, Risk**
  Identified/classified “secrets” ACs harvest (HWIDs, process memory, user paths) and performed risk/threat modelling to balance privacy vs. security gains.
- **Week 3 – Humans, Measuring**
  Treated humans as the weakest link in confidentiality: AC engineers shipping over-privileged drivers, users blindly accepting EULAs, and operators mishandling telemetry. I quantified the human-triggered leakage surface (e.g., what gets collected when users run with admin rights) to “measure” that fragility instead of just machine metrics.
- **Week 4 – Insiders, Confidentiality**
  Showed how kernel-privileged ACs act like insiders breaching confidentiality boundaries; proposed least-privilege and transparency recommendations.
- **Week 5 – Privacy, Integrity**
  Focused on privacy violations and data integrity (unauthorised memory reads); redacted logs responsibly and analysed the impact of integrity-check bypasses.
- **Week 7 – Data, Authentication**
  Explored data access and handle validation: hooked `NtOpenProcess` / `NtReadVirtualMemory`, and demonstrated privilege misuse via handle lateral movement.
- **Week 8 – Accidents, PKI**
  Discussed how unsigned drivers / ValidSection bypasses expose PKI gaps and “accident” risks; documented safeguards to keep research out of production harm.
- **Week 9 – Communication, Protocols, High Impact Risks**
  Communicated high-impact risks through a concise video/report; suggested protocol/API changes

vendors could adopt to make telemetry auditable and privacy-aware.

---

9. Future Work

- Signed minifilter for FS monitoring without kdmapper side-effects.
- Longer, scheduled captures to observe periodic telemetry.
- Automated dashboard (ELK/Grafana) for privacy event visualisation.

---

10. AI Assistance Declaration

Portions of wording/structure in this document were polished with a generative AI assistant. All code, logs, and technical analysis are mine. No fabricated references or results were introduced.

Appendix Index (Evidence Map)

- **E1** KDMapper MDL path source diff (`kdmapper-master/kdmapper/main.cpp`)
- **E2** ValidSection bypass code (`HelloWorld/callbacks.cpp`)
- **E3** HelloWorld driver core files (`Driver.c`, `Hooks.cpp`)
- **E4** DebugView / WinDbg session logs (See the video below)
- **E5** Redacted `BattlEye.log` & `EasyAntiCheat.log` + legend
- **E6** Demo video link (unlisted YouTube)
- **E7** Original Technical Guide Doc (README.md)

(End of main deliverable. Appendices in github.)