BESS Indigenous Endpoint Detection & Response (EDR) Prototype

Full Project Report

---

1. Project Title

BESS – Bharat Endpoint Security System (Indigenous EDR Prototype)

---

2. Introduction

BESS is an indigenous Endpoint Detection & Response (EDR) prototype designed to ensure that endpoint security, telemetry, policies, and update mechanisms remain inside national infrastructure. The project emphasizes **data sovereignty**, **offline-first operation**, and **cryptographically assured updates**, serving as an academic demonstration of how a lightweight EDR can function without reliance on foreign cloud platforms.

This project aims to develop: - A C++-based endpoint agent - A Flask-based Command & Management Server (CMS) - Secure communication using Mutual TLS (mTLS) - A cryptographically signed update system - Basic rule-based host intrusion alerts - Local SQLite storage with WAL mode

The project focuses **only on user-space implementation**, excluding kernel-level drivers, making it feasible for academic development.

---

3. Problem Statement

Most EDR solutions in India rely on foreign-developed cloud infrastructures. This creates: - Data sovereignty concerns - Dependency on foreign security tools - Lack of local control over updates & telemetry

Therefore, a minimal *indigenous* EDR prototype is needed to demonstrate: - Local storage of telemetry - Local update system - Lightweight on-device threat detection - Fully controlled on-premise infrastructure

---

4. Objectives

Technical Objectives:

1. Develop a **lightweight C++ endpoint agent** that collects security telemetry.
2. Build a **Flask-based CMS** for device registration, telemetry ingestion, policy distribution, and update management.
3. Implement **mutual TLS authentication** between agent and server.
4. Design a **secure, signed update pipeline** using RSA/ECDSA signatures.
5. Create a **basic behavioural detection rule engine**, e.g., detecting suspicious process execution paths.
6. Store telemetry in **SQLite (WAL mode)** with optimized indexing.

Academic Objectives:

- Demonstrate core principles of EDR systems.
- Provide a foundation for future kernel-level expansion.
- Showcase a fully indigenous cybersecurity prototype.

---

5. Feasibility Study

Technical Feasibility

Feasible Components:

- C++ user-mode service for file hash scanning, process monitoring.
- Flask REST API server with authentication.
- SQLite database for lightweight telemetry storage.
- Secure communication using certificates.
- Digital signature verification.
- Rule-based detection engine.

All these components require skills that are achievable at the academic level.

Non-Feasible (Out of Scope for Students):

- Kernel drivers (Windows/Linux)
- Advanced behavioural analytics (AI-based)
- Large-scale distributed cloud infrastructure
- WHQL code signing for Windows drivers
- Commercial-grade threat intelligence integration

Thus, the project is **highly feasible** as a prototype.

Financial Feasibility

The estimated budget provided is **₹50,000**. This is sufficient for an academic prototype.

A cost breakdown is provided later in Section 10.

Operational Feasibility

The system can run on: - Basic Linux server (CMS) - Any Linux/Windows system (Agent)

System operation is user-friendly through: - A dashboard/CLI - Auto-registration and update mechanism

Schedule Feasibility

Expected duration: **10–12 weeks**. Timeline is provided in Section 11.

---

6. System Architecture

Overall Flow:

1. Endpoint agent collects telemetry → stores locally → sends batches to CMS.
2. CMS receives telemetry → stores in SQLite → shows in dashboard.
3. Policies are pulled from CMS → applied on agent.
4. CMS signs updates → agent verifies & installs.
5. Alerts generated based on behaviour rules.

Modules:

1. Endpoint Agent (C++)

- File hash scanning
- Process monitoring
- Telemetry batching
- Local caching (SQLite)
- Rule engine
- Update installer

2. Command & Management Server (Flask)

- Device registration API
- Policy API
- Telemetry ingestion API
- Update distribution
- Certificate validation

- Dashboard

3. Security Components

- mTLS between agent & CMS
- RSA/ECDSA signing mechanism
- Hash verification
- Rate-limiting & schema validation

4. Storage System

- SQLite tables:
- devices
- telemetry
- policies
- updates
- signatures

---

7. Detailed Module Description

7.1 C++ Agent (User Mode)

Responsibilities:

- Monitor running processes using `/proc` (Linux)
- Monitor suspicious file paths
- Compute SHA256 hashes of executables
- Store telemetry in SQLite (WAL mode)
- Batch send telemetry to CMS using HTTPS + client certificate
- Apply behavioural rules
- Download & verify updates
- Apply updates securely

Behavioural Rule Example:

If a process executes from:

```
/tmp
/var/tmp
/home/user/.cache
```

Trigger alert: **"Suspicious process execution path"**

7.2 Flask CMS

API Endpoints:

- `/register` → Registers new devices
- `/telemetry` → Accepts telemetry JSON
- `/policies` → Returns policies to agents
- `/updates` → Returns available update metadata/payload

Database Functions:

- Insert telemetry with timestamps
- Store device details with certificate thumbprint
- Store update metadata and signatures

7.3 Signed Update Pipeline

CMS Steps: 1. Package update 2. Generate SHA256 3. Sign using RSA private key 4. Push to CMS update table

Agent Steps: 1. Download package 2. Verify signature 3. Verify hash 4. Install update using atomic rename 5. Write rollback marker if failed

---

8. Project Novelty

Unique Features:

- Fully **offline-first** design
- Complete **local sovereignty** of telemetry
- Cryptographically auditable update system
- Indigenous EDR prototype (academic level)

Even though EDR itself is not new, creating a fully local and verifiable prototype is academically innovative.

---

9. Expected Outcomes

- Working C++ endpoint agent
- Working Flask server with secure certificate-based authentication
- Digital signature-based update system
- Behaviour rule engine
- Local SQLite telemetry storage
- Project report + demo

10. Budget Breakdown (₹50,000)

Item	Description	Cost (₹)
1. Development Laptop & Tools	Temporary workstation, utilities	12,000
2. SSL Certificates / PKI Setup	Demo CA, certificate generation	2,000
3. Cloud/Server Machine (Optional)	For hosting CMS (3 months)	6,000
4. Software Libraries & Dependencies	Crypto libs, Flask plugins	3,000
5. Testing Environment Setup	Virtual machines, test servers	5,000
6. Contingency & Maintenance	Unexpected costs	5,000
7. Documentation & Printing	Project final report, diagrams	2,000
8. Research & Learning Material	Books, courses	5,000
9. Developer Stipend (Team)	For development time & effort	10,000

Total Cost = ₹50,000

11. Project Timeline (12 Weeks)

Week	Task
1	Requirement gathering, architecture design
2	Setup CMS skeleton, SQLite schema
3	Implement device registration API
4	Implement telemetry API & storage
5	Build C++ agent core structure
6	Add process/file monitoring in agent
7	Implement secure mTLS communication
8	Create signed update pipeline (CMS side)
9	Implement update verification (Agent side)
10	Add behavioural rules engine
11	Dashboard / CLI for CMS

Week	Task
12	Testing, documentation, final report

12. Conclusion

BESS presents a practical, academically strong indigenous EDR prototype. It focuses on real-world concepts like endpoint monitoring, cryptographic security, policy management, and secure update distribution, but keeps implementation within realistic academic limits.

The project demonstrates mastery over: - C++ system programming - Flask backend development - Database handling - Secure communication - Cryptographic update distribution

This provides both educational value and a potential stepping stone for future advanced EDR development.

13. References

- SQLite Documentation
- Flask REST API Guidelines
- OpenSSL Certificate Generation Manuals
- NIST Guidelines on Secure Update Mechanisms
- Hashing & Signature Algorithms (SHA256, RSA, ECDSA)

End of Report