PROJECT DOCUMENTATION

Advanced USB & Mobile Endpoint Monitoring Framework

USB Device Control & Security Auditing System

File: `Advance_USB_MTP.py` | Language: **Python 3.13** | Platform: **Windows**

Document Type: Technical Project Documentation
Author: Shuvo Paul(UMID18112570874) (Cybersecurity Intern)
Project Category: Cybersecurity / Endpoint Protection / Blue Team
Framework: USB Device Control & Monitoring Framework
Date: February 28, 2026
Classification: Academic / Internal Use

USB Mass Storage Detection	MTP Mobile Device Tracking	File Transfer Auditing
Real-time monitoring of drive connections with hardware fingerprinting.	Captures Android & iOS devices invisible to standard tools.	Snapshot-based DLP with bidirectional copy/modify/delete detection.

Table of Contents

1 Project Overview

The Advanced USB & Mobile Endpoint Monitoring Framework is a Windows-based desktop security application developed entirely in Python. It provides real-time detection, monitoring, and forensic auditing of all external device activity — covering both USB Mass Storage drives and MTP-based mobile devices such as Android smartphones and iOS tablets.

The tool is structured as a lightweight, zero-dependency GUI application built on Python's standard Tkinter library. Three independent background daemon threads run concurrently and continuously to ensure no device event or file operation goes unrecorded.

Core Security Functions: USB detection · MTP mobile tracking · File transfer auditing · Forensic report generation

2 Practical Motivation

USB-based threats remain among the most prevalent vectors for data exfiltration and insider attacks in enterprise environments. The following real-world scenarios motivate this framework:

Threat Vector	Attack Description	Framework Response
BadUSB / USB Worms	Malicious firmware in USB devices executes code or keystroke injections on plug-in	Immediate connection alert with device hardware ID logged
Data Exfiltration	Employees copy confidential files to personal flash drives	Every file copied to USB is detected and logged in File Activity tab
MTP Blind Spot	Smartphone transfers bypass monitoring tools (no drive letter assigned)	Dedicated MTP loop queries PnP entities, not drive letters
Rogue USB Implants	Hardware implants masquerade as HID or storage devices	VID/PID/serial fingerprinting flags unrecognised device identities

Why MTP Matters: Standard monitoring tools detect USB by enumerating drive letters (C:, D:, E:). Smartphones connected via MTP never receive a drive letter — they are completely invisible to drive-enumeration approaches. This framework specifically patches this security gap.

3 Project Objectives

Each objective from the project specification is fully implemented in the codebase:

#	Objective	Implementation
1	Detect USB plug/unplug events in real time	usb_loop() — polls WMIC every 2 seconds, diffs drive sets
2	Extract device hardware identifiers	get_usb_info() — parses VID, PID, serial from USBSTOR PnP path
3	Track MTP mobile device connections	mtp_loop() — queries Win32_PnPEntity with keyword filtering
4	Audit file transfers to/from USB drives	file_activity_loop() with classify_usb_changes() + detect_usb_to_system()

#	Objective	Implementation
5	Generate timestamped audit reports	generate_report() — structured 4-section .txt report saved to disk

4 Practical Scope of the Project

A. Unauthorized USB Detection

The **usb_loop()** method executes on a daemon thread, issuing the following WMIC command every 2 seconds:

```
wmic logicaldisk get caption, drivetype
```

It compares the resulting set of removable drives (DriveType=2) against the previous poll's snapshot stored in **self.prev_drives**. All newly detected drive letters trigger a call to **get_usb_info()**, which issues a second WMIC query:

```
wmic diskdrive where "InterfaceType='USB'" get PNPDeviceID
```

The returned PnP device path is parsed to extract three hardware identifiers:

- Vendor ID (VID) — identifies the USB device manufacturer
- Product ID (PID) — identifies the specific device model
- Serial Number — parsed from the trailing segment of the USBSTOR path

B. MTP Mobile Device Detection

MTP devices are captured via the **mtp_loop()** thread, which queries all USB-connected PnP entities and filters against a manufacturer keyword list covering 23 major brands:

```
MTP_KEYWORDS = [
    "mtp", "android", "portable", "samsung", "xiaomi", "poco",
    "apple", "google", "vivo", "oppo", "oneplus", "motorola",
    "realme", "iqoo", "nothing", "honor", "tecno", "infinix",
    "asus", "sony", "huawei", "hmd", "lava"
]
```

Coverage: 23 smartphone brands are covered, including the entire Android ecosystem and Apple iOS. Any new device matching one of these keywords is immediately added to the Mobile (MTP) tab and the event log.

C. File Movement Auditing

The file auditing subsystem uses a snapshot-diffing architecture. Rather than installing kernel-level filesystem hooks, it takes lightweight periodic snapshots of each monitored directory and compares them to detect changes.

Method	Responsibility
snapshot_files(base)	Recursively walks a directory tree; records each file as path → (size, mtime) fingerprint
classify_usb_changes(old, new, drive)	Diffs two USB snapshots to classify MODIFIED, RENAMED, SYSTEM→USB COPY, and DELETED events
detect_usb_to_system(old_sys,	Identifies files that moved from USB to the host by matching

Method	Responsibility
new_sys, usb_snap, drive)	size+mtime across both filesystems
file_activity_loop()	Orchestration loop: re-snapshots every 3 seconds, calls both classifiers, updates stored snapshots

The rename detection algorithm is particularly notable: it identifies a rename operation when a file disappears from the old snapshot (by path) but a new file appears in the same snapshot with identical metadata (size and mtime). This allows accurate RENAMED detection without access to kernel-level filesystem events.

D. Reporting & Alerting Module

Every event across all three monitoring threads is routed through the **log()** method, which prepends a full ISO-format timestamp and appends the entry to the Logs tab text widget.

The **generate_report()** method produces a structured plain-text audit report saved with a timestamped filename (**Audit_Report_YYYY-MM-DD_HH-MM-SS.txt**). The report is divided into four sections:

- Section 1 — USB Device History: drive letter, VID:PID, serial, status, and timestamp for every device
- Section 2 — Connected MTP Devices: all mobile devices detected during the session
- Section 3 — File Activity Log: all classified file events with type, path, location, and time
- Section 4 — Raw System Logs: the complete chronological event stream

5 Tools & Technologies Used

Category	Technology	Role in Codebase
Language	Python 3.x	Core application — no external packages required
GUI Framework	Tkinter / ttk	5-tab desktop UI: Treeview tables, Listbox, Text log, dialogs
USB Monitoring	subprocess + WMIC	Real-time drive enumeration and PnP device queries on Windows
MTP Detection	Win32_PnPEntity via WMIC	Detects mobile devices without drive letters via keyword filtering
File Auditing	os.walk + os.stat	Lightweight snapshot fingerprinting using file size and mtime
Concurrency	threading.Thread (daemon)	3 independent loops running in parallel without blocking the UI
Reporting	Python built-in file I/O	Timestamped structured plain-text audit report generation
Device Fingerprinting	USBSTOR PnP path parsing	Extracts VID, PID, serial from Windows PnP device path strings

6 Practical Techniques Implemented

Security Engineering Techniques

- Hardware Event Monitoring — Polling-based real-time USB connect/disconnect detection at 2-second intervals using WMIC subsystem queries.
- Device Fingerprinting — Extraction of Vendor ID, Product ID, and serial number from USBSTOR PnP device paths enables unique per-device identification, supporting blocklist enforcement.
- Snapshot-Based File Auditing — Lightweight hash-free file fingerprinting using size and mtime avoids the need for kernel hooks while still detecting copy, modify, rename, and delete operations.
- Bidirectional Transfer Detection — Both System→USB and USB→System data flows are independently monitored, covering both exfiltration and ingestion scenarios.
- MTP Gap Coverage — Detection of mobile devices that bypass drive-letter enumeration closes a significant blind spot in standard endpoint monitoring tools.

Blue Team Techniques

- Insider Activity Tracking — Every file operation performed on a USB-connected drive is classified with event type, file path, location, and timestamp — creating an evidence trail.
- Unauthorized Hardware Detection — Any USB device insertion is immediately logged with its hardware identifiers (VID:PID and serial), enabling retrospective investigation.
- Data Loss Prevention (DLP) — The File Activity tab functions as a software DLP layer, flagging large or suspicious data movements between system directories and external devices.
- Forensic Reporting — Structured audit reports include all session data in a format suitable for incident investigation and compliance review.
- Endpoint Hardening Support — The framework produces the monitoring and logging infrastructure necessary to support USB policy enforcement on Windows endpoints.

7 Workflow & Architecture

Thread Architecture

The framework's concurrency model deploys three independent daemon threads, each owning a distinct monitoring domain and polling frequency:

Thread	Entry Point	Poll Interval	Monitoring Domain
USB Monitor	usb_loop()	2 seconds	Removable drive connect/disconnect; VID/PID/serial extraction
MTP Monitor	mtp_loop()	3 seconds	Mobile device connect/disconnect via PnP keyword matching
File Auditor	file_activity_loop()	3 seconds	File system snapshot diffing across USB drives and system dirs

Startup Sequence

1. __init__() is called — application state (sets, dicts, snapshots) is initialized.
2. snapshot_system() is executed immediately — capturing the baseline state of Desktop, Downloads, and Documents directories.
3. build_ui() constructs the 5-tab Tkinter interface (USB Devices, Mobile MTP, File Activity, Logs, Help).
4. Three daemon threads are launched: usb_loop, mtp_loop, file_activity_loop.

5. root.mainloop() blocks the main thread while the three daemons operate continuously in the background.

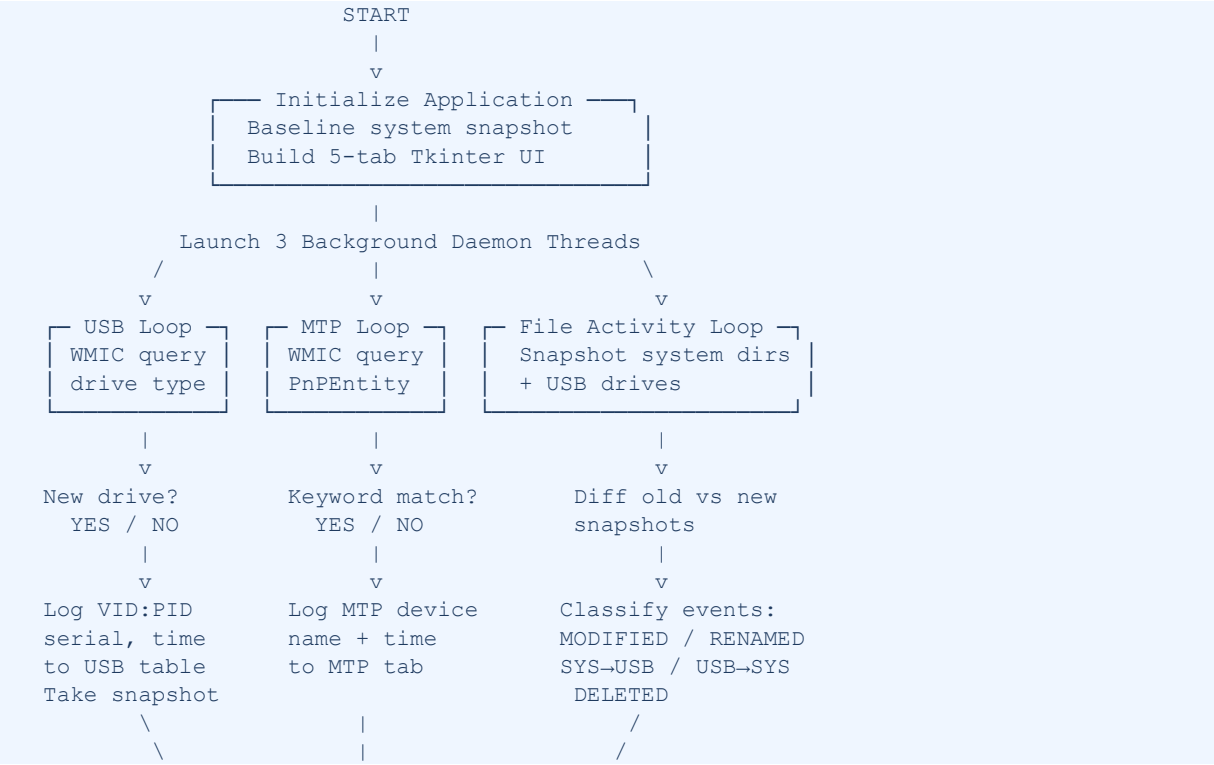
Per-Iteration Monitoring Logic

```
USB Thread (every 2s):
  ├── WMIC: enumerate drives with DriveType=2
  ├── diff(current_drives, prev_drives)
  ├── on new drive → get_usb_info() → log VID:PID:Serial → take initial snapshot
  └── on removed drive → clear usb_snapshot[drive] → log removal

MTP Thread (every 3s):
  ├── WMIC: query Win32_PnPEntity where DeviceID like 'USB%'
  ├── filter lines against MTP_KEYWORDS (case-insensitive)
  ├── on new match → append to MTP listbox → log connection
  └── on removed match → log disconnection

File Activity Thread (every 3s):
  ├── new_sys = snapshot_system()  [Desktop, Downloads, Documents]
  ├── for each active USB drive:
  │     new_usb = snapshot_files(drive)
  │     classify_usb_changes(old_usb, new_usb, drive)
  │       ├── MODIFIED:      file exists in both; size or mtime changed
  │       ├── RENAMED:       file removed + new file with same metadata
  │       ├── SYSTEM→USB:    new file on USB not in old_usb
  │       └── DELETED:       file in old_usb no longer present
  │     detect_usb_to_system(old_sys, new_sys, old_usb, drive)
  │       └── USB→SYSTEM:    new sys file matches old USB file by size+mtime
  │     usb_snapshots[drive] = new_usb
  └── system_snapshot = new_sys
```

8 Process Flowchart



```
             v              v                  v
          ┌──────── All events → Logs Tab ────────┐
          │  [timestamp] EVENT → file [location]  │
          └───────────────────────────────────────┘
                              |
                [User: Generate Report]
                              |
                              v
          ┌─────── Audit_Report_<timestamp>.txt ───────┐
          │  1. USB Device History                     │
          │  2. Connected MTP Devices                  │
          │  3. File Activity Log                      │
          │  4. Raw System Logs                        │
          └────────────────────────────────────────────┘
                              |
                             END
```

9 Expected Practical Output

USB Devices Tab — Sample Entries

```
Drive   VID:PID        Serial                     Status       Time
─────────────────────────────────────────────────────────────────────
E:\     0781:5583      4C530001261104104583        CONNECTED    14:32:07
F:\     058F:6387      058F63870000               CONNECTED    15:10:44
```

Mobile (MTP) Tab — Sample Entries

```
samsung android mtp device
apple mobile device (mtp)
xiaomi mi 11 (mtp device)
oneplus 9 pro (mtp)
```

File Activity Tab — Sample Events

```
Event                Time      Location  File
─────────────────────────────────────────────────────────────────────────────────
SYSTEM -> USB COPY   14:32:15  E:\       C:\Users\User\Documents\confidential.pdf
MODIFIED (USB)       14:33:02  E:\       E:\notes.txt
USB -> SYSTEM COPY   14:35:11  E:\       E:\photo.jpg → C:\Users\User\Desktop\photo.jpg
RENAMED (USB)        14:36:20  E:\       E:\old_name.docx → E:\new_name.docx
DELETED (USB)        14:37:44  E:\       E:\temp_file.zip
```

Generated Audit Report Structure

```
====================================================
ENDPOINT AUDIT REPORT - 2025-07-10 14:40:32
====================================================

--- 1. USB DEVICE HISTORY ---
Drive      VID:PID         Serial                         Status         Time
E:\        0781:5583       4C530001261104104583           CONNECTED      14:32:07

--- 2. CONNECTED MTP DEVICES ---
- samsung android mtp device

--- 3. FILE ACTIVITY LOG ---
```

```
Event                Time            Location        File
SYSTEM -> USB COPY   14:32:15        E:\             C:\Users\...\confidential.pdf

--- 4. RAW SYSTEM LOGS ---
[2025-07-10 14:32:07] USB INSERTED → E:\
[2025-07-10 14:32:15] SYSTEM -> USB COPY → C:\Users\...\confidential.pdf [E:\]
```

10 Code Structure Reference

Method / Component	Category	Description
MTP_KEYWORDS	Config	List of 23 smartphone brand keywords used for MTP PnP filtering
SYSTEM_DIRS	Config	Three monitored system directories: Desktop, Downloads, Documents
__init__()	Initialization	Sets up application state, takes initial snapshot, builds UI, launches threads
build_ui() / build_*_tab()	UI Layer	Constructs all 5 Tkinter tabs: USB, MTP, File Activity, Logs, Help
log(msg)	Logging	Prepends ISO timestamp and appends entry to the Logs panel
set_report_directory()	Reporting	File dialog for selecting the output folder for audit reports
generate_report()	Reporting	Writes structured 4-section audit report with timestamp in filename
snapshot_files(base)	File Auditing	Recursively fingerprints a directory tree as path → (size, mtime)
snapshot_system()	File Auditing	Aggregates snapshots across all configured system directories
classify_usb_changes()	File Auditing	Diffs two USB snapshots; classifies MODIFIED / RENAMED / COPY / DELETED
detect_usb_to_system()	File Auditing	Detects metadata-matched files transferred from USB to host system
record_file_event()	File Auditing	Inserts event row into File Activity table and writes to log
file_activity_loop()	Thread	Primary file auditing loop; re-snapshots and diffs every 3 seconds
get_usb_info()	USB Monitor	Extracts VID, PID, and serial number from WMIC USBSTOR output
usb_loop()	Thread	USB drive connect/disconnect detection loop; polls every 2 seconds
mtp_loop()	Thread	MTP mobile device detection loop; keyword-filters PnP entities every 3 seconds

11 Learning Outcomes

This project delivers practical, hands-on experience across multiple domains of cybersecurity and systems engineering:

- USB communication and Windows OS internals — understanding how WMI/WMIC, PnP device paths, and USBSTOR identifiers work at the system level.
- Attacker techniques — how BadUSB, USB worms, and MTP mobile devices are used for malware delivery and unauthorized data transfer.
- MTP monitoring gap — understanding why mobile devices evade traditional monitoring and how PnP entity-level detection closes this gap.
- Snapshot-based file auditing — implementing a lightweight DLP mechanism using periodic file system snapshots as a practical alternative to kernel hooks.
- Multithreaded application design — building concurrent Python applications using daemon threads with shared mutable state.
- Forensic reporting — generating structured, evidence-quality audit logs suitable for incident response and compliance review.
- Blue Team endpoint security engineering — building a functional monitoring toolkit using only Python's standard library.

12 Project Deliverables

#	Deliverable	Description	Status
1	Advance_USB_MTP.py	Fully functional USB & MTP monitoring toolkit — 377 lines of Python	Complete
2	Project Documentation	This document — architecture, code analysis, workflow, and output reference	Complete
3	Audit Reports	Audit_Report_<timestamp>.txt files generated by the tool during operation	Generated at runtime
4	Flowchart	Process diagram included in Section 8 of this document	Complete
5	Final Presentation	PPT slide deck to be prepared based on this documentation	Pending

End of Documentation — USB Device Control & Monitoring Framework
`Advance_USB_MTP.py | Python 3.13 | Windows`