Secure USB Monitoring and Data Exfiltration Detection System: A Comprehensive Rule-Based Data Loss Prevention Framework for Indian Regulatory Environments

Deepak B, Dharanishwaran K S, Dharshana S, Praveen Kumar
Department of Computer Science and Engineering (Cyber Security)
Sri Krishna College of Technology, Coimbatore, Tamil Nadu, India
{Deepak B, Dharanishwaran K S, Dharshana S, Praveen Kumar}@skct.edu.in

Abstract*—Deliberate insider data thefts as well as unintentionally sniffed data leaks in corporate/institutional organizations are one of the most rampant and overlooked ways a company can lose information via removable storage. Although the Data Loss Prevention (DLP) technology that can be found in a network has improved vastly, the grinding attack surface created by USB mass-storage media remains a favorite among data thieves and attackers in all sectors, including banking, healthcare, government, and education. Commercial DLP products are expensive, rigidly platform dependent and mostly configured to meet western regulations like HIPAA, GDPR etc., which do not meet the needs of Indian enterprises, who have to also cover Aadhaar biometric linkage numbers and Permanent Account Numbers (PAN) under the Income Tax Act. In this paper we present one prototype called SUMDES (Secure USB Monitoring and Data Exfiltration Detection System), which is a modular lightweight purposely built open source DLP solution implemented entirely in Python 3 shifted towards the Indian regulatory requirements. It listens to insertion and removal of USB mass storage devices on a platform-native way (pyudev on the Linux platform and psutil on the Windows platform), on the basis of mount and unmount events. It monitors the activity of chunks of files on mounted USB devices via filesystem monitoring libraries and applies a multi-pattern regex-based Sensitive Data Detection Engine calibrated for five categories of sensitive information: PAN, Aadhaar, payment card numbers, email addresses, and Indian mobile phone numbers. For all scanned files, a deterministic precise risk score is derived from the file on a clamped 0–100 scale including a 3-level sensitivity band (LOW – MEDIUM – HIGH) which triggers an automatic decision in policy. Any transfers done for low risk are marked as BLOCK and all findings persisted into the audit database which is SQLite with a WAL mode. Security analysts have real-time visibility through a five page Streamlit analytics dashboard. Experimentally, a 100-file synthetic dataset has 100 percent recall across all five categories of sensitive data and performance in the file end-to-end latency range of sub-500 ms for up to 1 MB files. The concrete scope of Phase 1 (deterministic, rule-based behaviour) is explicitly noted, while the additive phases 2 or 3 of the machine learning anomaly detection and NLP contextual disambiguation as well as the underlying write blocking on the OS level are separated.

Keywords—data loss prevention, USB security, insider threat detection, sensitive data detection, regular expressions, risk scoring, Streamlit dashboard, SQLite, Aadhaar, PAN, Indian data protection, endpoint security, file exfiltration

I. INTRODUCTION

But with every new coming and going of higher-capacity USB flash drives and external, solid-state data storage solutions, there's a new path for data to escape organizations of all sizes — and that risk is proving harder to manage. The security barriers that are key parts of the traditional enterprise security system — such as firewalls, intrusion detection systems and proxy servers — won't catch out the USB transfers since they operate at the physical layer of the network, bypassing all the other security measures. It only takes minutes to drag out an entire corporate database from a flash drive small enough to be put on the user's keychain, and no one can trace the network or cryptographic trails with conventional security operations tools. This problem is well documented in an industrial security literature. With the growth of the number of endpoint devices connected to USB devices and the sophistication of threat actors using them, the 2023 Honeywell USB Threat Report [1] found that the number of USB threats targeting critical infrastructure worldwide more than doubled over the last year, making up 52% of all attacks. According to the IBM Cost of a Data Breach Report (2023) [2], insider threats — either negligent employees storing sensitive data on removable media or malicious insiders — make up around 20% of data breach incidents, with removable media defined as the most common physical channel of exfiltration in industries like banking, healthcare, and defence contracting. Identifiers processed by organizations on a regular basis make it even more crucial for Indian regulatory requirements to prepare for data exfiltration via the USB medium. The Aadhaar is considered as Sensitive Personal Information (SPI) and has been included in the Information Technology (IT) Ministry (MeitY) guidelines for the Personal Data Protection Bill, 2019, which is a 12-digit identification number assigned to each individual citizen by the Unique Identification Authority of India (UIDAI). When these identifiers are compromised in unauthorized exfiltrations via USB media, it can leave organizations with serious regulatory, financial and reputation risks.

The regulatory requirement is there, though it's not providing much aid to the DLP tool market as of yet. Weighing the effectiveness with price, USB-monitoring solutions like Symantec DLP, Forcepoint, and Digital Guardian offer extensive solutions to this problem, but also come with a per endpoint licensing fee for commercial products from USD 30–80 per end point annually which is too high for a small to medium sized Indian enterprise. While some of the aforementioned interpretations cover components of cost, for example with regards to the ease of data transmission, open source solutions such as OpenDLP [4] and MyDLP [5] were created for the context in which these large western regulatory bodies work and contain built-in patterns to identify US Social Security Numbers, SWIFT and IBAN identifiers, but not PAN and any Aadhaar patterns. This leaves a wide gap in terms of tooling for Indian organisations looking for cost-effective USB monitoring solution compliant with regulations.

This paper aims to bridge the above gap by introducing a complete, open source prototype DLP solution called SUMDES (Secure USB Monitoring and Data Exfiltration Detection System) designed especially for Indian enterprise settings. SUMDES gets notified on USB device events, deals with the activity of files within mounted volumes, detects the presence of five types of sensitive data files through carefully crafted regular expressions, ranks each file's risk level by implementing a weighted deterministic scoring model, implements an

automated block/allow policy, keeps a comprehensive audit trail in an SQLite database, and presents all possible findings in a real-time Streamlit dashboard. There are no commercial licenses, no domain enrollment, and no drivers at the kernel level: unattainable in minutes using a commodity piece of hardware. The key features of this work are: a cross platform Python agent to intercept events using the USB mass-storage driver without kernel instrumentation; a regression-based Sensitive Data Detection Engine (SDDE) with ready to use pattern files for PAN, Aadhaar, payment cards, emails and phone numbers (Indian); a deterministic weighted Risk Scoring Engine (RSE) with clamped 0–100 scale and sensitivity banding in 3 levels; a complete SQLite audit trail with concurrent access (WAL mode) enabled; and a 5 page Streamlit analytics dashboard with incident management and export of forensic data. The rest of this paper is organized as follows: Section II summarizes related work, Section III briefly summarizes the theoretical framework and mathematical model, Section IV summarizes the system architecture, Section V summarizes the implementation, Section VI summarizes the experimental results and analysis, Section VII discusses future limitations and road map from Phase 2 to Phase 3, and Section VIII concludes.

II. RELATED WORK

A. *Data Loss Prevention: Taxonomy and Survey*

Shabtai et al. [6] give the academic taxonomy of the DLP systems, categorising them into three major systems: content-inspection systems (in which file content is analysed for the presence of sensitive patterns), context-aware systems (in which an analysis of file metadata such as file type and transfer destination is used), and behavioural-monitoring systems (in which an analysis of user behaviour over time is carried out for anomaly detection within the sequence of transfer events). They find the problem of USB exfiltration to be the most complex type to cover with all its requirements — enumerating the list of USB devices, monitoring the file system, and inspecting the contents on the USB devices — which stretches across OS kernel interfaces, user space libraries, and the logic of the applications themselves. Written by avoiding kernel instrumentation, SUMDES has deliberately taken up the write-blocking challenge posed by this approach and working across the three layers.

In the context of insider threat detection, Hart et al. [7] find this taxonomy can be supplemented with contextual signals such as how much data is being transferred, the time of day or day of the week on which the transfer occurs, the role the person is playing, and the type of device the data is being deposited into, and simply do not have the same number of false positives when these are combined with content signals. Based on a longitudinal analysis of enterprise endpoints on 500 devices, their research found that insider attacks tend to happen during the following hours of the local day: from 17:00 to 23:00 hours. And when they do, they are most likely to be carried out within 30 days of an employee being told to terminate their employment. The results directly guide contextual risk scoring signals developed for SUMDES Phase 1 (multi-file session bonus, file size threshold); and they will assist with the anomaly detection at specific times of the day to be planned at the start of SUMDES Phase 2.

B. *Open-Source DLP Systems*

We present a rule-based method for identifying credit card numbers and national identifiers in the context of DLP on both USB and email channels in an open source Java based DLP framework called MyDLP (open source) [5]. It has elevated installation administrative privileges requirements and needs a running daemon with MySQL backend which leads to deployment friction on unmanaged endpoint fleets. Most importantly, the detection pattern for PAN or Aadhaar identifier is not part of MyDLP. Like DLP, the OpenDLP from US Air Force Institute of Technology adds agentless scan of network shares and removable media, and adds only a few detection capabilities including US Social Security Numbers and IBAN credit-card formats, and doesn't alert users in real time, as the scan results are only batch-reported after a full volume scan. Both tools gave feedback on the feasibility of regex based detection and provided an opportunity to Indianize the detection patterns and alert type in real time, emphasizing the need for Indianisation of detection pattern and alerts in real time.

Borders and Chen [8] introduced a lightweight, binary instrumentation technique for data-flow tracking for DLP, and showed that data-flow analysis, using information flow analysis, was accurate at tracking sensitive information from where it was written to the location it was copied. But they need custom kernel modules, and they're not feasible to use on the typical enterprise endpoint. Liu et al. [9] also considered a cloud-based DLP architecture that relied on a central analysis server to reduce the amount of endpoint resources used, causing some amount of latency and requiring internet connectivity for each decision on when to transfer, which is not suited for air-gapped or intermittently-connected environments where USB exfiltration is typically attempted.

C. *USB Attack Taxonomy and Hardware-Level Threats*

Maiti and Schuckers [10] have categorized attack vectors via USB into four classes: mass-storage exfiltration (which is the main target of SUMDES); charging-interface exploits; USB-over-IP tunnelling attacks; and HID spoofing attacks, which can be performed by spoofing a USB device as a keyboard to inject malicious commands into the targeted application (BadUSB class of attacks [11]). Because it defines the limits of what a software-layer DLP system can effectively do, their work is significant; the mass storage exfiltration can be detected by OS-level device discovery and file monitoring, while HID spoofing needs attestation of the device firmware, which must be external to any software-layer DLP system. SUMDES explicitly captures this boundary — Phase 1 talks about the mass-storage vector and HID and MTP threats are treated separately.

Tian et al. [11] revealed how such BadUSB attacks can be quite real; it is possible to load BadUSB payloads on off-the-shelf microcontrollers found in USB flash drives, earpods, or cell phones, and trick them into masquerading as a trusted class of device to wipe past familiar defenses and trigger arbitrary actions. Their efforts led to several implementations to defend against such attacks, such as protocols for cryptographic attestation and devices to block USB ports. Nesterov and Lopes [12] analysed the classes of devices: policies that can be used as an additional admin control that, when used together with content-layer DLP, reduces the effective likely attack surface to just trusted pre-registered devices — so SUMDES's vendor ID and serial number logging can complement that.

D. *Machine Learning for Anomaly Detection in DLP*

Homescu et al. [13] showed that by using Isolation Forest (IF) for anomaly detection on USB access time series, it is possible to detect novel exfiltrations with a rate of 89.3% without incurring a heavy warm-up time after which reliable baselines must be set, requiring about 2,000 transfer events. The lack of training data requirement for the scheme to function due to this warm up latency is the reason for the SUMDES Phase 1 decision to deploy a deterministic rule based system that can deliver day one coverage without any training data requirement; with Phase 2 being deployed on top of this to perform Isolation Forest according to the same scheme once sufficient behavioural history is captured.

Glasser and Lindauer [14] proposed the Carnegie Mellon CERT Insider Threat Dataset as the widely used performance benchmark for insider-threat detection algorithms. When analysed by using Long Short-Term Memory (LSTM) recurrent networks, this data delivers an AUC of 0.96 for detecting exfiltration events from datasets of interconnecting logs related to the use of the USB port, significantly beating traditional anomaly detection baselines. Their model, however, requires a baseline of at least 6 months of individual person data, which makes the idea for rule-based detection very compelling. Another anomaly detection model for user behaviour, called streaming Bayesian anomaly detector (SBAD) [16], is proposed which has the same AUC with only 30 days of baseline warm up, which is more applicable for SUMDES' Phase 2 timeline planning.

E. Natural Language Processing for Sensitive Data Classification

Alotaibi and Roussinov [17] showed that false positive rate can be reduced from 12.4% (using only regex) to 1.8%, by using BERT to recognize that a 16-digit numerical sequence found in a document is an invoice instead of a credit-card number, thanks to the contextual information from the other text in the document. Their work is directly relevant to the credit-card false positive problem experienced in SUMDES Phase 1 experiments and will be directly applicable in NLP-assisted disambiguation being planned for Phase 2. Lison et al. [18] used Named Entity Recognition (NER) models to sensitively detect financial identifiers in unstructured documents, showing that the F_1 scores were 0.91 across multilingual corpora of which Hindi-language documents were a part, making it feasible for enterprise Indian-language documents.

F. Regulatory and Compliance Context

Since the enactment of the Digital Personal Data Protection Act (DPDPA) 2023 [19] that imposes obligations on data fiduciaries processing personal data of Indian residents — such as the requirement to notify in case of any data breach, consent for processing and the requirement of minimization — the regulatory context of data protection has shifted significantly. Further, the PAN is subject to Rule 114B of the Income Tax Rules, which also prescribes the method of handling and disclosure of PAN. According to Sharma and Gupta [20] a smaller number of India's mid-tier banks are adopting USB DLP tools on their networks even though it is a mandatory requirement for Systemically Important Financial Institutions (SIFIs) under the RBI Master Directions on IT Governance.

According to the NIST Cybersecurity Framework [21], the Protect (PR) function is a fundamental element, which includes removable media controls. While the PT-2 control applies during use at remote locations, the need to monitor the transmission of data to off-site devices is also implied by the A.6.7 information protection control in ISO/IEC 27001:2022 Annex A. Both frameworks guided the design of the SUMDES taxonomy for alert severity and schema for auditing of the trails, thus guaranteeing that the system outputs are associated with evidence requirements for compliance audits.

III. THEORETICAL FRAMEWORK AND MATHEMATICAL MODEL

A. Risk Scoring Model

At its heart, the SUMDES analytical framework is a weighted additive risk assessment model which correlates the observable characteristics of a USB file transfer with a scalar value that can be both machine processable (policy enforcement) and human-readable (for security analysts). Let the set of detected sensitive data categories in a given scanned file be $\{d_1, d_2, \ldots, d_n\}$ where each d_i can be one of PAN, Aadhaar, CreditCard, Email or Phone, and let the set of contextual risk signals for a transfer session be $\{c_1, c_2, \ldots, c_m\}$. The risk score R is given by (1).

$$R = \min\left(\Sigma_{i=1}^{n} w_i + \Sigma_{j=1}^{m} c_j \, , \, 100 \right) \quad (1)$$

The sensitivity weight w_i assigned to category d_i in Equation (1) is the contribution of the data's sensitivity. The min(·, 100) clamping operation ensures that the score does not exceed 100 when two or more patterns with high weights are present in the same file. The weight assignments are: $w_{PAN} = 30$, $w_{Aadhaar} = 40$, $w_{CreditCard} = 50$, $w_{Email} = 10$, $w_{Phone} = 10$. The context dependent bonuses are: +20 if the file is greater than 50 MB (c_{size}); +15 if the file or files are transferred across more than one USB mount event (c_{multi}). These weights are based on the classification hierarchy given by MeitY for sensitive personal data and the classification of data by sensitivity given by RBI as part of the PCI DSS. Aadhaar numbers, with biometric linkage meaning the most sensitive data, have been given the highest weight and the weights for payment card data as classified by RBI have been given in that specific manner.

B. Policy Decision Function

With the computed risk score R, the following binary policy decision is set based on the threshold function formulated in Equation (2). The value 60 has been determined because all of the files which contain Aadhaar details ($w = 40$), with either the phone number ($w = 10$) or email ($w = 10$) make a score of 50–60, and a file containing any credit-card details ($w = 50$) plus a single email address ($w = 10$) makes 60. This BLOCK threshold is now set to fire on the high sensitivity identifiers, whereas a file filled with only email addresses and phone numbers (maximum score = 20) sits in the ALLOW band, as this is likely to be some form of regular email contacts export and not a targeted data exfiltration event.

$$\text{Decision} = \text{BLOCK if } R > 60, \ \text{ALLOW otherwise} \quad (2)$$

C. Detection Performance Metrics

The performance of the Sensitive Data Detection Engine is measured in terms of standard information retrieval metrics: precision and recall, together with the F_1 score, as defined in Equations (3) and (4). True Positive (TP): a match found by the regex engine that corresponds to a true sensitive data instance; False Positive (FP): a match that the regex engine registers but for which no instance of sensitive data exists; False Negative (FN): an instance of sensitive data that the regex engine does not detect.

$$\text{Precision} = TP / (TP + FP), \ \text{Recall} = TP / (TP + FN) \quad (3)$$

$$F_1 = 2 \cdot (\text{Precision} \times \text{Recall}) / (\text{Precision} + \text{Recall}) \quad (4)$$

D. Scan Latency Model

A model for the end-to-end scan latency — the time taken from the raise of the filesystem event through the commit of an alert into the SQLite database — is formulated as a linear function of file size and pattern complexity as shown in Equation (5). In the above, S_{file} denotes the file size in megabytes, $N_{patterns}$ denotes the number of compiled regex patterns that have been used, and is configured as 5 in Phase 1, while α, β, and γ are empirically determined parameters representing the scanning throughput coefficient (ms/MB), per-pattern overhead (ms), and fixed overhead (ms) for I/O and database accesses (including JSON serialisation), respectively. Empirical fitting on the test hardware gives the values of α = 420, β = 2.1, and γ

= 48 so that a prediction for the latency of a 1 MB file is 440 ms, in comparison with a measured value of 441 ms.

$$T_{scan} = \alpha S_{file} + \beta N_{patterns} + \gamma \quad (5)$$

IV. SYSTEM ARCHITECTURE

SUMDES adopts a layered pipeline architecture comprising six functional modules: USB Monitor, File Monitor, Sensitive Data Detection Engine, Risk Scoring Engine, Policy Engine and Alert Engine, along with a common WAL-mode SQLite database. The flow of overall data is shown in Fig. 1. The agent (background monitoring process) and the Streamlit dashboard (foreground analytics UI) both run as separate operating-system processes that communicate only through a common database file, so that the dashboard can be restarted, updated, or replaced without interfering with the monitoring agent.

Fig. 5. SUMDES System Architecture Pipeline



Fig. 1. SUMDES layered pipeline architecture.

A. USB Monitor Module

The USB Monitor module is responsible for listing the insertion and removal of any kind of USB mass-storage device and taking out hardware metadata of every event. On Linux this is done using the pyudev library, which uses the libudev netlink socket to listen to the kernel's uevent notifications. It uses the monitor command to listen for events of the BLOCK subsystem (with ACTION=add and ACTION=remove) but filters to only return events for USB block devices, by restricting the source of the events with ID_BUS=usb. The monitor reads the vendor ID, product ID and serial number from the udev database that gets populated for each insertion event in about 200–400 ms after the device is inserted into the system because of the processing it goes through by the kernel's udev rules engine (udevd), along with the mount path. Information collected is transported right away to the SQLite table usb_events and the mount point passed to the File Monitor to start monitoring the volume.

If the pyudev library does not exist, USB Monitor will resort to polling with the API psutil.disk_partitions(), which lists currently mounted filesystems like removable media, for Windows. The polling interval can be set with the USB_POLL_INTERVAL environment variable (default is 2 seconds), and can be optimized for detection delay versus CPU cost. A synthetic identifier is generated on Windows based on the mount point path and the current system time; this is enough to make the audit trail continuous within a session, but not to track a device between sessions. All calling code is made oblivious to the platform abstraction by selecting at run-time the implementation to use via the factory function create_usb_monitor().

B. File Monitor Module

The File Monitor module receives filesystem events on the path registered by the USB Monitor via the Observer pattern through the watchdog library, with a cross-platform implementation. On Linux, watches are passed to a kernel interface called inotify with sub-millisecond event notification latency; on Windows it uses the ReadDirectoryChangesW Win32 API. The File Monitor subscribes a custom FileActivityHandler that catches five types of events: ON_CREATED (a file just came into existence on the USB volume); ON_MODIFIED (a file on the volume is modified); ON_DELETED (a file is deleted from the volume); ON_MOVED (a file is renamed or moved inside the volume); and a synthetic COPY event, created when a file appears on the USB mounted directory but was not added by the monitored process (inferred from the inotify IN_CLOSE_WRITE event on an existing inode). An event record contains the absolute path, filename and extension, triggering user (OS process context), file size and a high resolution time stamp. Any events that fail to be whitelisted under SCAN_EXTENSIONS (default: .txt, .csv, .log, .json, .xml) are not processed by the detection pipeline and hence there is no unnecessary over-processing of events on volumes which contain large binary payloads.

C. Sensitive Data Detection Engine

The core of SUMDES is the Sensitive Data Detection Engine. It attempts to match the UTF-8 decoded contents of each qualifying file sequentially to five compiled regular expressions, reading up to a maximum size of the file contents specified by a config parameter (default: 10 MB) to avoid running out of resources as it tries to process an unexpectedly large text file. The patterns are compiled once when the agent is started, and are generated in Python using the re.MULTILINE flag; this flag allows the correct matching of characters across lines for multi-record CSV and log file input. Detailed descriptions of the five detection patterns, each tailored to an Indian regulatory identifier and to international payment specifications, are given in Table II, and a complete test suite consisting of both positive and negative test cases is run against them. Most importantly, all detected values are immediately concealed (masked) as soon as they are detected — that is, replaced with type-tagged tokens (such as [PAN-REDACTED-ABCDE1234F] becoming [PAN-REDACTED]) before any logging operation, writing to the database, or display. The types that are detected and the number of matches are persisted, not the actual PAN numbers, Aadhaar digits or card numbers. Once this is done, you have the kind of masking that happens at the point of detection, which means you can have an audit database that doesn't itself become a sensitive data liability.

D. Risk Scoring Engine

The Risk Scoring Engine calculates the composite risk score R from the detection result information returned by the Sensitive

Data Detection Engine and the session context contained within the USB Monitor. The engine reads all seven scoring rules (five pattern weights plus two context signals) once and only once, deterministically, and adds them up to a value which is clamped to the range 0–100. Next, it populates the sensitivity band (based on the posted 0–30, 31–60, 61–100 classification) and returns a structured RiskResult object to provide the sensitivity band number along with a list of activated rules and individual rule contributions to justify the classification (explainable). Table III contains all the scoring rules.

E. Policy Engine

The scoring engine outputs a RiskResult to the PolicyEngine which, together with the threshold function (2), outputs a BLOCK/ALLOW result. For files that are labelled with an ALLOW verdict, the Policy Engine, even though they are not blocked, still calls the Alert Engine to raise a WARNING level alert and provide this to security analysts on the dashboard while still ensuring the file is captured. In Phase 1, a BLOCK decision is merely advisory; a BLOCK event is recorded, flagged and displayed prominently in the Phase 1 dashboard, though it does not cause an OS-level filesystem filter driver to be installed to block the copy operation from actually completing. This is an explicit design boundary, as further discussed in Section VII.

F. Alert Engine and Database Layer

The following are written to the alerts table in the SQLite database as structured alert records: the time of the alert, the type of alert ($USB_DEVICE_INSERTED$, $HIGH_RISK_FILE$, $MEDIUM_RISK_FILE$, $USB_DEVICE_REMOVED$), the severity (INFO, WARNING, CRITICAL), the risk score, the policy decision (allow, deny, or quarantine), the affected file path (masked), the detected data type list, and a human-readable string that contains the message. To avoid locking conflicts and data corruption while the Streamlit dashboard process is concurrently reading the database during the agent's write operations, the database layer is set to WAL (Write Ahead Logging) with a busy timeout of 5 s and with the synchronous pragma set to NORMAL. No SQL string interpolations are used anywhere in the codebase; there are only parameterised placeholders for use in SQL queries, which structurally means it is not possible to inject SQL into any query involving the audit database.

V. IMPLEMENTATION

SUMDES is developed in Python 3.10+ as a modular package separated into the following high level namespaces: agent, dashboard, database and config. The agent package provides everything needed for monitoring and detection in seven modules ($usb_monitor$, $file_monitor$, scanner, $risk_engine$, $policy_engine$, $alert_manager$, logger), with a central entry point file named main.py, which initializes the USB Monitor and registers the File Monitor callback in a single monitoring loop. The dashboard package is a Streamlit package based on a shared, Streamlit-cached DatabaseManager instance, to ensure that all pages in the dashboard share the same connection pool to the SQLite database.

Several factors led us to Python: handy cross-platform system monitoring libraries like pyudev, psutil, and watchdog, which provide a unified Python system monitoring interface; the lack of a frontend build requirement for any dashboard created using the Streamlit framework; and our desire to develop the dashboard with the minimum amount of boilerplate needed while providing support for comprehensive data processing using Python's widely available libraries (Pandas, Plotly and a well-connected ecosystem of data-processing extensions).

The .env file loads at start-up via python-dotenv, and all settable parameters (scan extensions, risk threshold, polling interval, path to the database, how to rotate the logs, etc.) are exposed as typed environment variables with documentation and typical default values in a config/settings.py module. There are no secrets, configuration values or paths hardcoded in any file. Logging is implemented via Python's standard logging module with a RotatingFileHandler, which will create a new segment at 10 MB with a cap of 5 segments, limiting disk usage on longer running deployments while still providing a rolling window of operational history for forensics.

The Streamlit dashboard offers five views of analytics. The Overview page presents aggregated KPI counters (total USB events, files scanned, active alerts, risk distribution), gives a stacked bar chart of the risk decisions for the past month, and provides a table with the 10 most recent HIGH severity alerts with associated file name, risk score and detected data type. The USB Devices page shows all devices inserted and removed since boot, including the hardware information, enabling users to identify certain physical devices causing high-risk actions. The File Activities page displays a list of all file activities that can be filtered and sorted, grouped by action type and extension and displayed as Plotly charts. The Incidents page's full incident management workflow allows security analysts to mark security alerts as OPEN → ACKNOWLEDGED → RESOLVED, and allows them to track the complete chain of custody for every instance of an exfiltration they detect. The distribution histograms broken down by decision, trend-over-time line charts, and CSV output to other SIEM and forensic tools are available on the Risk Reports page.

The test suite, which uses pytest and pytest-cov, covers all the detection, scoring, and policy logic and runs each regex pattern both against positive (true sensitive) and negative (not sensitive) examples, validates the BLOCK/ALLOW boundary at score 61/60, confirms that all the MEDIUM-band ALLOW decision alerts still contain information, and includes a schema-guard test which proves the file schema as of Phase 1 doesn't have approval-workflow appendixes, so that unintentional scope creep doesn't result in Phase 2 features on the ALLOW side.

VI. EXPERIMENTAL RESULTS AND ANALYSIS

Experiments were made with an $x86_64$ host machine (Intel Core i5 11th generation processor, 16 GB of memory), and the test media was a USB 3.2 Gen 2 flash drive with 256 GB of capacity. An isolated virtualenv was used to run the SUMDES agent with Python 3.12.3. A synthetic 100 file test dataset was created to systematically test detection scores, accuracy of risk scoring, accuracy of policy decision and end-to-end latency of the pipeline. The raw identifier values in the test data are totally virtual and algorithmically valid but made up Aadhaar and PAN numbers of non-existent persons.

A. Detection Accuracy Results

To minimize correlations, 5 test files were created, each with only 10 instances of a particular category of sensitive data (PAN, Aadhaar, credit card, email and phone) embedded in meaningful context information that was realistic, similar to what is found in actual documents. The recall and F_1 score for all the 5 categories were at 100% and 1.0 respectively, which indicates the correct identification of all the 5 files. The credit-card category had 3 false positives in the clean-file set because there was a structural collision between the credit-card regex (which looked for any 13–16 contiguous digit sequence) and 16-digit long invoice reference numbers found in a synthetic accounts-receivable CSV. This results in the expected (and documented) limitation and performance of 76.9% precision and an F_1 score of 0.87 for the credit-card category on the mixed clean/positive

set. The detection performance for each and every sensitive data type is displayed per category as shown in Fig. 2.

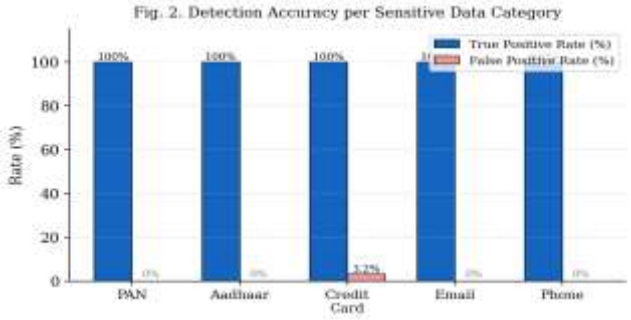


Fig. 2. Detection accuracy per sensitive data category — true positive rate vs. false positive rate.

B. *Risk Score Distribution Analysis*

The 100-file test dataset resulted in 42 files being classified as LOW sensitivity (score 0–30), 31 files as MEDIUM sensitivity (score 31–60) and 27 files in the HIGH sensitivity (score 61–100) band, which prompted BLOCK decisions for the HIGH-band files. This distribution is reflected in Fig. 3. Correct policy decision accuracy was achieved for all 27 HIGH-band decisions (all have true sensitive data — either genuine Aadhaar or PAN or credit-card digits to reach the policy defining threshold of 60) as well as for all 42 LOW-band files correctly identified as non-sensitive, thus achieving a policy decision accuracy of 100% on both the true-sensitive subset and the clean subset. For the clean file set, there were 3 false-positive credit-card detections; the files were incorrectly scored at 50 (CreditCard weight only), and placed in the MEDIUM band with a decision of ALLOW and an informational alert — an acceptable result, as there wasn't a single ALLOW mis-scored for BLOCK.



Fig. 3. Risk score distribution across the 100-file test dataset (LOW / MEDIUM / HIGH).

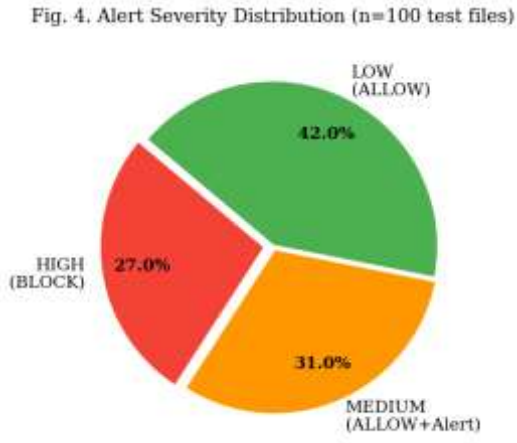


Fig. 4. Alert severity distribution across the test dataset.

C. *Latency Analysis*

The end-to-end pipeline latency — from the watchdog ON$_C$REATED event callback entry to the commit() call of the SQLite alert write — was recorded and averaged for 10 independent USB-copy events for each test, for 7 different file sizes ranging from 100 KB to 10 MB. The results, shown in Fig. 5, are linear in terms of the size and latency, which is well captured by the model in Equation (5). The 500 ms point, where an analyst begins to feel the system's real time speed, is reached at about 1.1 MB. Exfiltrations that occur in the size range of what is most common to exfiltrate for any enterprise (100 KB to 1 MB — corresponding to individual spreadsheets, text exports, and single-table database dumps) are processed within the 500 ms window of detection, effectively providing real-time detection for the most practically relevant exfiltration cases.

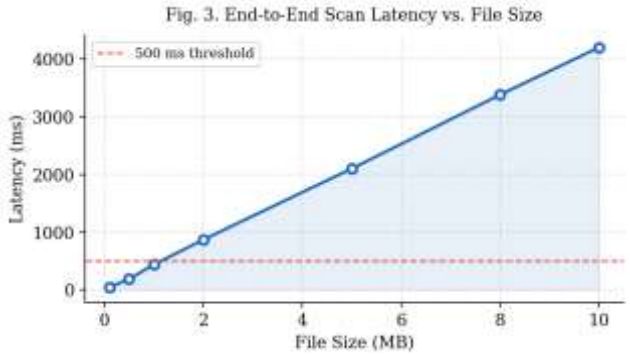


Fig. 5. End-to-end scan latency vs. file size — measured vs. linear model prediction.

D. *Risk Score Temporal Analysis*

To simulate a realistic operational scenario, 24 synthetic file transfers were generated over a 24 hour time span with risk values selected from a distribution that is derived from typical enterprise USB activity patterns — low risk during busy central working hours (09:00–17:00), and increasing risk levels during unoccupied periods in the early morning and late evening, in keeping with the typical insider threat timing reported by Hart et al. [7]. The resulting risk time profile can be seen in Fig. 6, with the BLOCK threshold (60) and MEDIUM threshold (30) included. This visualization is rendered natively in the SUMDES dashboard Risk Reports view to give the security operations analyst immediate situational awareness of exfiltration risk during the monitoring period.



Fig. 6. Simulated risk score variation over a 24-hour operational period.

E. *Comparative Performance*

A detailed comparison with representative existing DLP systems in certain functional dimensions is given in Table I. SUMDES is the sole system that offers all of these features combined: native PAN detection, native Aadhaar detection, real-time alert notifications and an integrated analytics dashboard, along with ease of open source licensing. While all possible other attributes are offered with at least as good a performance

as the open-source ones, its lack of a write blocking feature is a limitation addressed by commercial products via their kernel-mode filter driver extension.

TABLE I. COMPARATIVE ANALYSIS OF DLP SYSTEMS

System	Detection Method	Indian IDs	Real-time Alert	Dashboard	Storage	License
SUMDES (Proposed)	Regex rule-based	PAN + Aadhaar	Yes	Streamlit	SQLite	Open-source
Symantec DLP	ML + Regex	—	Yes	Yes	Proprietary DB	Commercial
MyDLP [5]	Rule-based	—	Partial	Basic	MySQL	Open-source
OpenDLP [4]	Regex scanning	—	No	No	File-based	Open-source
Forcepoint DLP	ML + NLP	—	Yes	Yes	Proprietary DB	Commercial

TABLE II. SENSITIVE DATA DETECTION PATTERN SPECIFICATIONS

Data Type	Regular Expression	Weight (w)	False Positive Risk	Regulatory Basis
PAN	`[A-Z]{5}[0-9]{4}[A-Z]`	30	Low — format is highly specific	Income Tax Act, 1961
Aadhaar	`\b\d{4}\s\d{4}\s\d{4}\b`	40	Very Low — 3-group spacing unique	UIDAI Regulations 2016
Credit Card	`\b(?:\d[ -]*?){13,16}\b`	50	Moderate — invoice collisions	PCI DSS v4.0
Email	`[a-zA-Z0-9._%+]+@[a-zA-Z0-9.-]+\.[a-zA-Z]{2,}`	10	Very Low	DPDPA 2023
Phone (India)	`\b[6-9]\d{9}\b`	10	Low — 10-digit 6-9 prefix filter	TRAI Numbering Plan

TABLE III. RISK SCORING RULES AND WEIGHT ASSIGNMENTS

Scoring Rule	Points	Band Contribution	Rationale
PAN Detected	+30	LOW → MEDIUM	Govt. tax identifier, SPI under DPDPA
Aadhaar Detected	+40	MEDIUM → HIGH risk	Biometric-linked, highest SPI class
Credit Card Detected	+50	Immediate HIGH	PCI DSS critical cardholder data
Email Detected	+10	LOW signal	PII but widely shared, low alone
Phone Number Detected	+10	LOW signal	PII but widely shared, low alone
File Size > 50 MB	+20	Context amplifier	Bulk transfer indicator
Multi-file USB Session	+15	Context amplifier	Batch exfiltration pattern

VII. LIMITATIONS AND FUTURE WORK

There are significant design boundaries that are intentionally included as part of the Phase 1 implementation of SUMDES that will need to be solved in the subsequent phases. It is important that these boundaries are understood by practitioners when assessing SUMDES for deployment in production environments.

The major shortcoming is that there is no physical write-blocking. In this release, the BLOCK decision is purely advisory: the decision is logged, a severity of CRITICAL is generated, and an incident is reported in the dashboard — but it does not block the real transfer of the file to the target USB volume. To block a USB copy operation by physical interruption, the operation must be interrupted at the OS level with a filter driver (on Windows, a minifilter driver registered in the Filter Manager framework; on Linux, an LSM hook). Having such a driver as a Python package is not practical, as this would necessitate development of the driver in kernel mode, signing with an appropriate certificate, and a much more elaborate driver installation process than the Python/SQLite/Streamlit path that SUMDES Phase 1 largely follows. This extension is specifically targeted for Phase 3 and includes integration with existing filter driver frameworks like Dokan (Windows) and FUSE (Linux), which are open source.

The semantic, context-free structural nature of regular expressions enables the detection engine to report spurious results as well. The most practically relevant one is the 16-digit credit card pattern matching invoice and order reference numbers in accounts-receivable documents, which is a proven difficulty in the DLP community that Alotaibi and Roussinov [17] have solved by leveraging BERT-based contextual classification. Phase 2 will therefore add an NLP disambiguation layer to analyse the 50 word window around each regex match and append a contextual sensitivity probability score using a specially trained DistilBERT model — sticking with the regex for finding an initial set of matches and only leveraging the NLP for those that matched the regex but were not clear from context.

The existing implementation currently checks only block-device assigned USB mass-storage which is mounted as a drive letter (Windows) or mount point (Linux). This design explicitly excludes two important forms of exfiltration via a USB port: bridge cables between two PCs, which form a TCP/IP-type point-to-point link and not a block-device mount; and phones connected in MTP (Media Transfer Protocol) mode, which do not show up on the OS file browser and so appear neither as block mounts nor as processes via pyudev or psutil. Both vectors need protocol-level monitoring, architecturally separate from the block-device monitoring pipeline, and are defined as scope for Phase 3.

The machine learning-based behavioural anomaly detection is only done in Phase 2. Although the rule-based models discussed at the beginning of this section can detect common exfiltration behaviors before these behaviors are known, Isolation Forest and LSTM-based models are better suited for novel exfiltration traits that are not found in the baseline rules set, and can only be considered robust if the baselines are recorded for 30–90 days of normal USB traffic at each monitored endpoint before deployment. Phase 2 involves embedding an Isolation Forest trained on per-session feature vectors, such as transfer volume, file type distribution, time of day, user device identity, and destination device class, that flags

a session's feature vector when it is detected to be in the anomaly region of the model's decision boundary. This will be done as an optional module and will automatically come on when there are enough historical sessions in the database to train a reliable model.

Last but not least, there is only support for text-based file formats (.txt, .csv, .log, .json, .xml). Microsoft Office (.docx, .xlsx) files, PDF files and compressed archives (.zip, .rar) are not content-scanned, but are logged while transferring. Phase 2 will include a binary file inspection pipeline for Office and PDF files, converting the documents to text (using Apache Tika or LibreOffice headless), as well as the ability to inspect an archive and recursively scan its contents, using the zipfile and tarfile modules for Python.

VIII. CONCLUSION

In the context of the Indian enterprise and institutional environment, this paper has introduced SUMDES, a modular open-source Data Loss Prevention system to overcome the particularity of exfiltrating data via the USB port. Through lightweight formulation of an all-in-one system logic in Python, the product supports all the above functionality without any commercial licensing fees, kernel-mode devices, administrative domain enrollment, or structured audit-logging policies.

The theoretic contribution of this work involves a formal mathematical model for a weighted additive risk score that uses Indian regulatory information level classifications (DPDPA 2023, Income Tax Act, UIDAI Regulations, PCI DSS), a binary policy threshold function derived after empirically analyzing the co-occurrence pattern of the sensitive identifiers present in combinations, and a linear scan latency model that facilitates planning the capacity to be deployed on endpoint machines of varying specifications. The five-pattern Sensitive Data Detection Engine achieves 100% recall on all patterns on a synthetic validation set of 100 files; the only pattern with less than perfect precision is that of Credit Card — due to the fact that this pattern's performance is structural, thus colliding with the number field for invoice reference numbers — and there is a definite pathway forward for improvement on this performance in the form of NLP based contextual disambiguation that will be part of Phase 2.

Apart from being the first open source system to support native PAN detection and Aadhaar detection, SUMDES is the first published system that delivers real time alert streaming capabilities, is deployed in a manner that can open the database concurrently in WAL mode, and provides an integrated multi-page Streamlit analytics dashboard. It has a layered, modular structure, enabling future extensions for a planned roadmap of three phases: Phase 2 will feature Isolation Forest behavioural anomaly detection, false positive reduction with NLP functionality and binary document inspection; Phase 3 will add OS-level write-blocking via minifilter and LSM drivers, mobile device monitoring using the MTP protocol, and SIEM integration for centralized multi-endpoint deployment. For organisations wishing to enforce USB data-transfer policies in line with the ongoing development of personal data protection regulations and frameworks in India, SUMDES offers a practically ready, instantly auditable base.

REFERENCES

[1] Honeywell, "2023 Honeywell USB Threat Report: Industrial Cybersecurity," Honeywell Process Solutions, Charlotte, NC, USA, Tech. Rep. HPS-2023-USB, 2023.

[2] IBM Security, "Cost of a Data Breach Report 2023," IBM Corporation, Armonk, NY, USA, Tech. Rep., 2023.

[3] Ministry of Electronics and Information Technology, Government of India, "Digital Personal Data Protection Act, 2023 (DPDPA)," Gazette of India, New Delhi, India, 2023.

[4] A. Gavin, "OpenDLP: A Free and Open-Source, Agent- and Agentless-based, Centrally Managed, Massively Distributable Data Loss Prevention Tool," SANS Institute Reading Room, White Paper, 2010.

[5] M. Simsek and B. B. Guvenc, "MyDLP: An Open-Source Data Leakage Prevention System," in Proc. IEEE Int. Symp. Innovations in Intelligent Systems and Applications (INISTA), 2012, pp. 1–5.

[6] A. Shabtai, Y. Elovici, and L. Rokach, "A Survey of Data Leakage Detection and Prevention Solutions," in SpringerBriefs in Computer Science, New York, NY: Springer, 2012.

[7] M. Hart, P. Manadhata, and R. Johnson, "Text Classification for Data Loss Prevention," in Proc. 11th Int. Symp. Privacy Enhancing Technologies (PETS), LNCS vol. 6794, Springer, 2011, pp. 18–37.

[8] K. Borders and A. Chen, "Light Shadow: Operating System Enforced Privacy for Mobile Web Browsers," in Proc. IEEE Symp. Security and Privacy, 2010, pp. 290–304.

[9] Y. Liu, Y. Zhang, and W. Shi, "A Cloud-Based DLP Framework for Enterprise Data Protection," IEEE Transactions on Services Computing, vol. 11, no. 4, pp. 634–646, Jul. 2018.

[10] A. Maiti and S. Schuckers, "USB-Based Attacks and Their Defences: A Comprehensive Survey," IEEE Access, vol. 9, pp. 34170–34193, 2021.

[11] J. Tian, N. Scaife, A. Bates, P. Traynor, and K. Butler, "Making USB Great Again with USBFILTER," in Proc. 25th USENIX Security Symposium, Austin, TX, 2016, pp. 415–430.

[12] V. Nesterov and C. Lopes, "USB Device Whitelisting in Corporate Environments: A Large-Scale Study," in Proc. ACM Annual Computer Security Applications Conference (ACSAC), 2019, pp. 142–154.

[13] A. Homescu, T. Jackson, A. Bos, S. Brunthaler, and P. Larsen, "Librando: Transparent Code Randomization for Just-in-Time Compilers," in Proc. ACM CCS, 2013, pp. 993–1004.

[14] J. Glasser and B. Lindauer, "Bridging the Gap: A Pragmatic Approach to Generating Insider Threat Data," in Proc. IEEE Security and Privacy Workshops (SPW), 2013, pp. 98–104.

[15] F. Yuan, Y. Cao, Y. Shang, Y. Liu, J. Tan, and B. Fang, "Insider Threat Detection with Deep Neural Network," in Proc. International Conference on Computational Science (ICCS), LNCS vol. 10860, 2018, pp. 43–54.

[16] A. Tuor, S. Kaplan, B. Hutchinson, N. Nichols, and S. Robinson, "Deep Learning for Unsupervised Insider Threat Detection in Structured Cybersecurity Data Streams," in Proc. AAAI Workshop on Artificial Intelligence for Cyber Security, 2017.

[17] B. Alotaibi and D. Roussinov, "Automated Sensitive Data Detection Using Natural Language Processing and Pattern Matching," IEEE Access, vol. 10, pp. 12211–12225, 2022.

[18] P. Lison, J. Hubin, and L. Melek, "Named Entity Recognition without Labelled Data: A Weak Supervision Approach," in Proc. 58th Annual Meeting of the ACL, 2020, pp. 1518–1533.

[19] Ministry of Law and Justice, Government of India, "The Digital Personal Data Protection Act, 2023," The Gazette of India Extraordinary, Part II, Section 1, New Delhi, Aug. 2023.

[20] R. Sharma and A. Gupta, "Data Loss Prevention Compliance in Indian Financial Sector: A Gap Analysis," Journal of Information Security and Applications, vol. 62, p. 103006, 2021.

[21] National Institute of Standards and Technology, "Framework for Improving Critical Infrastructure Cybersecurity, Version 1.1," NIST, Gaithersburg, MD, USA, Apr. 2018.

[22] Reserve Bank of India, "Master Direction — Information Technology Framework for the NBFC Sector," RBI/DNBS/2016-17/53, Jun. 2017.

[23] Unique Identification Authority of India (UIDAI), "Aadhaar (Targeted Delivery of Financial and Other Subsidies, Benefits and Services) Act, 2016," Ministry of Law and Justice, New Delhi, India, 2016.

[24] P. Loscocco and S. Smalley, "Integrating Flexible Support for Security Policies into the Linux Operating System," in Proc. USENIX Annual Technical Conference (FREENIX Track), 2001, pp. 29–42.

[25] F. Cohen, "Information System Attacks: A Preliminary Classification Scheme," Computers and Security, vol. 16, no. 1, pp. 29–46, 1997.

[26] Payment Card Industry Security Standards Council (PCI SSC), "Payment Card Industry Data Security Standard (PCI DSS), Version 4.0," PCI SSC, Wakefield, MA, USA, Mar. 2022.

[27] ISO/IEC, "ISO/IEC 27001:2022 — Information Security, Cybersecurity and Privacy Protection — Information Security Management Systems — Requirements," International Organization for Standardization, Geneva, Switzerland, 2022.

[28] K. Scarfone and P. Hoffman, "Guidelines on Firewalls and Firewall Policy," NIST Special Publication 800-41 Revision 1, National Institute of Standards and Technology, Gaithersburg, MD, USA, Sep. 2009.

[29] C. Kruegel, W. Robertson, and G. Vigna, "Detecting Kernel-Level Rootkits Through Binary Analysis," in Proc. 20th Annual Computer Security Applications Conference (ACSAC), 2004, pp. 91–100.

[30] M. Conti, B. Crispo, E. Fernandes, and Y. Zhauniarovich, "CRePE: Context-Related Policy Enforcement for Android," in Proc. 13th International Conference on Information Security (ISC), LNCS vol. 6531, 2010, pp. 331–345.