Enhancing USB Security: A Multi-Layered Framework for Detecting Vulnerabilities and Mitigating BadUSB Attacks

***Abstract*—Universal Serial Bus (USB) devices are an inseparable part of modern computing, and they are also a considerable cybersecurity threat. Malicious hardware and BadUSBs Keyloggers and other malicious peripherals that have been reprogrammed can pose as legitimate devices, execute commands that are not authorised, and steal confidential data without the user's knowledge. This thesis introduces a detailed software-based system that attempts to identify and prevent malicious USB actions by employing a multi-layered security system. The proposed system incorporates USB metadata validation system, behavioral tracking, anomaly detection, and user-verification to offer high protection without affecting usability. The framework is deployed by the device connection and it temporarily isolates the device as it tries to verify the authenticity of the device by analysing power consumption, keystroke timing, and CAPTCHA and mouse hover-based user authentication. In its simplest form, a machine-learning model trained on real and GAN-enhanced data would allow the process of adaptive threat detection that can detect changing attack patterns with high accuracy. Experimental tests prove that the system attains high detection effectiveness and a very low false-positive level, which effectively eliminates the risks of malicious USB devices. This product fills a severe gap in endpoint protection by providing a practical, smart and user-friendly solution to protect personal and enterprise space against the emerging threat of USB-based attacks.**

***Index Terms*—BadUSB attack, Anomaly detection, Device fingerprinting, Forensic analysis, Behavioral analysis, Metadata validation, Adaptive learning**

I. INTRODUCTION

A. Background and Motivation

Universal Serial Bus (USB) devices have become an essential part of everyday computing and are used for data transfer, device communication, and peripheral control. With this widespread use, they have also become a common medium for carrying out security attacks. Reports suggest that around 25% of malware infections are now linked to USB devices, making them one of the most frequently exploited attack sources [1]. Attackers use this interface to deliver payloads, steal information, or execute hidden commands without user awareness.

Among these, one of the most severe categories is the BadUSB attack, where the internal firmware of a USB controller is reprogrammed to act maliciously while still appearing as a normal peripheral. Such devices can emulate trusted components like keyboards or network adapters to inject commands or spread malware, completely bypassing standard security checks [2]. Traditional antivirus systems and endpoint tools mostly inspect software processes or file activities, but they do not monitor the behavior of the hardware layer, allowing firmware-level threats to remain undetected.

Attackers also often hide their traces by modifying device descriptors or power signatures. Fake vendor IDs, mismatched serial information, and unusual power consumption are early signs of compromise, but most systems ignore them. Since most existing USB defense mechanisms focus on a single layer, either metadata or user confirmation, they fail to detect complex, multi-stage attacks that involve both behavioral and hardware-level manipulation.

To address these limitations, this research focuses on developing a complete software-based protection system capable of identifying and mitigating malicious USB devices through several coordinated stages. Our framework integrates USB metadata verification, behavioral analysis, power profiling, and user authentication into a single system that monitors device behavior in real time. This approach keeps normal usability intact while preventing harmful activities before execution. The motivation behind this work is to create a practical and adaptive security solution the USB Guard Framework, which can automatically detect, block, and report any unauthorized USB actions with minimal interference to the user.

B. Problem Statement

Universal Serial Bus (USB) devices have now become a common feature in daily computer use, enabling people to transfer files, add peripherals, and generally make things happen with simplicity. Such convenience, however, does not come without enormous security risks. Attackers have discovered methods of misusing the implicit trust that computers place in USB connections over the years. Most concerning of all is the rise of so-called BadUSB attacks in which a seemingly normal USB drive or keyboard is reprogrammed in a way that is undetectable to perform malicious tasks, including keylogging, infecting a system with malware, or, in some cases, gaining complete control of a system. The thing that is especially treacherous about these attacks is that they are stealthy. The majority of the common security controls, e.g., antivirus software or firewalls, are set up to identify threats on the software level. They will tend to miss the activity at the physical or behavioral layer, such as the minor variations in typing rates, unusual power consumption, or a device impersonating another device. Rogue USBs can therefore bypass security very easily and infect personal and organizational systems. Although developers and researchers have devised measures to assist in USB security tightening, the majority of available solutions lack a comprehensive real-time approach. The solution that is needed is one that is intelligent, multi-layered, and can not only detect the threat at an early stage but also one that can learn new forms of attacks, and also be simple to use. The study is aimed at bridging that gap. The key issue is that there is no decent system that would have capabilities to proactively analyze USB activity, learn normal patterns that indicate an attack, and defend users, without becoming obtrusive. Being capable of cracking it would result in a more secure computing world where USBs can be utilized without trepidation, particularly in security-sensitive environments such as health care, finance, and key infrastructure.

C. Research Objectives

The core goal of the proposed work is to design an effective and smart framework (especially for BadUSB-based attacks) that can identify, prevent, and neutralize malicious USB devices. In response to the growing threat of cyber-attacks through USB peripherals, the proposed research seeks to implement a multi-layered security mechanism containing real-time monitoring, behavioral analysis, USB metadata validation, and anomaly detection using machine learning.

1. Detect Malicious USB Devices: Design a detection layer that continuously monitors USB behaviour using both device information and user interaction. It will actively monitor features like device descriptors, metadata, keystroke delays, and more to identify anything unusual that may suggest malicious activity.

2. Mitigate USB Threats: Develop a multilayered security system that acts immediately when a malicious activity is detected. The system will automatically block, log, report, and unauthorize the device if any malicious activity takes place. It should also ensure that user tasks remain uninterrupted.

3. Maintain Usability and Accessibility: The framework should also be lightweight and user-friendly. A small CAPTCHA and mouse hover verification mechanism is used to confirm human user is present. This will reduce the chance of any automated or unauthorised access while ensuring easy user interaction.

4. Apply Machine Learning for Anomaly Detection: Machine learning models need a properly balanced dataset and training to learn to distinguish between malicious and non-malicious devices with the lowest possible false positives, so that it can easily detect suspicious activity. Active threat monitoring based on valid collected data should help improve detection accuracy, which should ultimately help the system to identify unseen or unrecognised attacks.

5. Protect System Integrity and Data: This framework should be suitable for securing any environments where USB attacks could cause major damage (e.g., labs, enterprises, industrial systems, hospitals, financial networks). The framework should allow non-malicious trusted devices without hampering workflow. Also, the framework should immediately stop unauthorized access or malicious devices. Which will preserve both safety and stability.

All in all, our work aims for a multilayer, adaptive defense mechanism that will continuously monitor and learn device behavior, verify user intent, as well as protect against USB-based attacks in real time without interrupting regular user activity.

D. Contributions and Scope

Our research contributes by providing a complete defense solution against USB-borne attacks. It uses a multilayer protection model which combines metadata validation, behavioral profiling, power consumption analysis, user verification, and more within a unified synchronous detection system. Traditional USB endpoint security solutions rely mostly on the application layer as well as specific signatures and hash-based defenses, which usually need continuous new updates in order to detect new threats. Our framework provides a smart, multilayered, adaptive defense that continuously learns and responds to USB behavior, providing user-friendly, affordable, and strong protection against modern USB threats.

The major contributions of this study are:

A Multi-Layer Security Framework
A unified multilayered framework that combines USB metadata verification, behavioral monitoring, and power consumption profiling as well as human-level verification such as CAPTCHA and mouse hover verification. This multilayered approach ensures early threat detection as well as prevention before a malicious device can execute any harmful payload or activity.

Real-Time Detection and Mitigation
Our frameworks leverage trained models to analyze forensic and behavioral anomaly detection. XGBoost gave us the highest accuracy among all(Logistic Regression, SVM, KNN, Decision Tree). All these models were trained on a balanced dataset which included real and CTGAN generated synthetic data to enhance the model's accuracy and robustness.

Dataset Development
We have built a custom dataset for our model training, for which we have collected 2,653 real USB connection records (both malicious and non-malicious) then we have used GAN-based augmentation to generate 4347 synthetic entries which made our final dataset of 7,000 entries. For malicious entries, we have collected and run different payloads on devices, extracted their data and behavior and later labeled them as malicious in our dataset. Our final dataset is balanced and improves the generalization and performance of the detection models.

Usability and System Adaptability
We have introduced a very simple yet effective human verifi-

cation layer that maintains usability while enduring security. It's a modular design that allows it to be adapted to personal, corporate, or industrial applications.

Scope

The framework is designed as a lightweight, software-only solution compatible with standard Linux systems. It covers threat detection from metadata inconsistencies, abnormal power behavior, keystroke anomalies, and non-human interaction patterns. The system is suitable for securing endpoints in general computing environments as well as in critical sectors such as healthcare, finance, and industrial control systems. Future versions may expand to cross-platform compatibility and integration with enterprise-level intrusion response systems.

We have designed the framework to be a multilayered, lightweight user friendly solution which is compatible with standard Linux systems. Our framework integrates USB metadata inconsistencies, behavioral analysis, power profiling, keystroke anomalies, and user authentication into a single system that monitors device behavior in real time. This approach keeps normal usability intact while preventing harmful activities before execution. The framework is suitable for securing USB endpoints in general computing environments as well as in critical sectors such as labs, healthcare, finance, and industrial control systems. We also aim to expand it to cross-platform compatibility as well as integration in enterprise-level intrusion response systems in future versions.

II. RELATED WORK

A. Existing USB Security Solutions

According to the prior studies, USB devices continue to remain one of the largest attack surfaces due to the lack of proper authentication and insecure firmware trust models, unable to provide sufficient protection [3], [4], [5]. The paper [4] identified that old protocol bugs made it possible to affect peripherals to impersonate themselves. Also, it enables any arbitrary code execution, which allows attackers to bypass OS-level trust. Besides, USB-Watch [6] introduced a small dongle that monitors USB traffic to allow it to immediately detect strange HID behaviour. Additionally, forensic log-correlation systems [7] scanned system events and keyboard keystrokes to determine when commands are being injected or re-injected. These studies have enlightened us to have a glimpse of what is occurring and provided some good forensic hints. But they still continue to fail to seal the scalability, cross-platform adaptability, and real-time defense that we are after.

B. Hardware- and Firmware-Level Defenses

FirmUSB [8] used symbolic execution at the firmware level to find malicious device endpoints and hidden device functionality in USB firmware. The research on BadUSB vulnerabilities [9], [10] proposed secure partitions, attestation of firmware, and verification of signature to prevent controller reprogramming. Spyduino [11] also showed that low-end microcontrollers were demonstrated to be able to impersonate trusted devices by explaining the ease with which firmware can be compromised. A recent work, USB-GATE [12], demonstrated gateway filtering and adversarially trained models to increase firmware validation. Although these techniques enhance firmware security and traceability, they require special hardware and complicated implementation procedures. Hence, they do not work well in a dynamic or enterprise environment.

C. Software and Behavioral Approaches

Software-based approaches is basically used for attempting to identify shady USB solutions by tracing the behavioral analysis and what is happening during the runtime. A paper regarding USB keypress injection detection [13] used the delay of the keystroke to determine whether it was a human typing or a bot. Another paper [14] applied CNNs and LSTMs (Long Short-Term Memory) to examine current traces and identify suspicious power consumption patterns. Broader studies on keyloggers [15] also outlined the strategies of detecting time patterns as a way to catch automated USB attacks. These methods achieve fairly high precision under controlled settings, but the problem is- they are generally one-dimensional and do not combine other hints such as metadata, power measures, or context of interaction.

D. Machine Learning in USB Anomaly Detection

Nowadays, Machine learning has become one of the primary facilitators of USB threat detection because it can detect dynamic threats by learning complex behavioral and forensic patterns that static rules cannot capture. Algorithms such as SVM, Random Forest, and XGBoost are useful to classify USB activities based on metadata, timing, and power features [13], [14], [16]. USB-GATE article [12] further advanced the machine learning direction based on GAN-generated data augmentation and transformer embeddings to enhance the resilience of models to adversarial attacks. Again, related work on power-anomaly detection [16] demonstrated that ML models can predict anomalous current traces in embedded systems reliably. Despite these amazing advances, the majority of the research is conducted on isolated areas: behavioral, power, or firmware, without integrating all of these in a holistic, context-sensitive detection framework.

E. Gap Identification

The existing practices did not offer a combined adaptive framework; rather, they target a single security aspect, such as hardware integrity, behavioral monitoring, power profiling, etc. Hardware-based defenses are firm and unadaptable, whereas software-based models are flexible and tend to overlook firmware-level trust and contextual information. Our proposed USB-Guard can fill this major gap by integrating forensic metadata validation, power profiling, and keystroke behavior analysis into a single, host-only application. Additionally, an extra user verification feature, CAPTCHA checking, can enhance protection against automated attacks without impacting usability and system performance.

III. PROPOSED FRAMEWORK AND IMPLEMENTATION

This part includes the proposed USB Protection Framework which combines multi-layered security elements to identify and stop malicious USB peripherals. The system is an integrated package of a strict hardware access control system, behavioral monitoring, anomaly detection through machine-learning and an adaptive interface to verify the user to ensure safe and authorized USB usage without losing usability. Subsections presented below discuss the architecture and implementation in detail.

A. Overall System Architecture

The framework follows a modular architecture which consists of five major components: (1) Strict Gate, (2) User Verification Layer, (3) Data Extraction and Profiling, (4) Machine Learning Analysis, and (5) Decision and Logging Module.

When a USB device is connected, the Linux udev subsystem triggers the Strict Gate that immediately puts the device in an unauthorized state in the /sys directory. Such a setup prevents the operating system to mount or execute the device without being authorised to do so. After isolating the device, the verification layer presents the user to CAPTCHA and mouse-hover challenges to verify that the user is interacting with the device. When the verification is successful, the system provides temporary authorization to the device and starts acquiring data using the PyUSB, psutil and evdev libraries. These are modules that fetch metadata descriptors, power consumption levels, CPU load and keystroke timing data. The obtained data are then converted into a feature vector that is organized and provided to a machine-learning engine that has been trained on authentic and GAN-enhanced data. The decision module will either approve or reject the device depending on the forecast made, and the result should be documented as proof of audit.

B. Strict Gate Mechanism

The first defense in the proposed system is the Strict Gate. It works on a kernel-user interface by monitoring the events of USB connection created by the udev subsystem. When a new USB insertion is detected, the system writes the flag authorized = 0 in the file /sys/bus/usb/devices/.../authorized and hence, the device is not initialized at the operating system level.

This gate will guarantee that no USB device will be able to communicate and execute payloads before passing verification and analysis. As an implementation, the mechanism was written in Python in the form of subprocess calls to the Linux sysfs interface. It acts as a managed isolation facility, which blocks unauthorized access and counters at the earliest opportunity possible an attack of the type known as BadUSB.

C. User Verification layer (CAPTCHA + Hover)

After the isolation of the device by the Strict Gate, the user is confronted with a minimal graphical user interface (based on the Tkinter Python), featuring a hover-over system and a captcha test. CAPTCHA prevents automated exploits or script activity related to devices operated by a robot whereas the hover validation ensures that human interface is met in real-time before a device can be authorized.

This method effectively addresses a common lapse of the traditional operating-system trust models whereby USB peripherals are automatically mounted without the user looking at them. In the event of either of the verifications, the system leaves the device in an unauthorized condition and logs the failure as a future threat. When the successful one is attained, the gate provisionally sets authorized=1 to access the examined controlled data.

D. Machine Learning Layer

The intelligence of the framework is that the Machine Learning Layer classifies every USB event as either benign or malicious. Supervised learning models are used in the system where a dataset of real USB traces is used to learn the models together with artificial samples created with CTGAN to obtain a balanced distribution of classes. The most prominent models include the Logistic Regression, Support Vector Machine (SVM), K-Nearest Neighbors (KNN), Decision Tree (DT), and XGBoost, which are implemented with the help of scikit-learn. USB metadata (Vendor ID, Product ID, Serial number, Power consumption, etc.), host process statistics and behavioural cues (e.g. keystroke timing) are stored as feature vectors. XGBoost model was the best in terms of accuracy and F1-score thus showing a high level of robustness in terms of differentiating between normal and malicious behaviour.

Every prediction gives a probability value that is used to determine the final decision of the system and thresholds can be adjusted to be more restrictive or liberal in their policies.

E. Metadata and Power Profiling Layer

This module focuses on low level USB descriptors and host side telemetry to identify anomalies that are of malicious device characteristics. The metadata parameters consist of: vendor id, product id, manufacturer, interface classification and serial number that is cross checked with a whitelist of trustworthy devices.

At the same time, data on power consumption are gathered through psutil and tracked to unexpected peaks which can be interrelated with some hidden processes like keystroke injection or firmware running. These parameters form an early behavioral fingerprint which is used in rule-based filtering as well as machine-learning inference features.

The system is able to detect devices that are reprogrammed or spoofed and identify them by a deviation in behavior rather than basing the identification on identifiers only.

F. End-to-End Workflow

The complete workflow proceeds as follows:

1. USB Plug-In: At first, after inserting a USB, there will be some event that occurs in Linux, and the computer will be aware that something is going on.

2. Strict Gate Activation: The OS puts the device into the unauthorized state of /sys which prevents any communication to occur until the user provides the correct keystroke.

3. User Check: There is a Tkinter pop-up window that contains a small CAPTCHA and a floating check that attests that the user is not a bot.

4. Data Extraction: After passing these tests, the PyUSB, psutil and evdev starts to grab USB descriptors, power usage and key-press timing off the stick.

5. Construction of Feature Vector: All that information is converted into a package in terms of a vector that fits with the learning model anticipated.

6. Machine Learning Analysis: The model makes the decision whether the device is non-malicious or threatening.

7. Decision Implementation: The system makes decisions on the device if the device is malicious or non malicious and takes actions like giving access or blocking.

G. Algorithm Design

The Figure 1 illustrates the hybrid detection pipeline of the proposed USB protection framework.

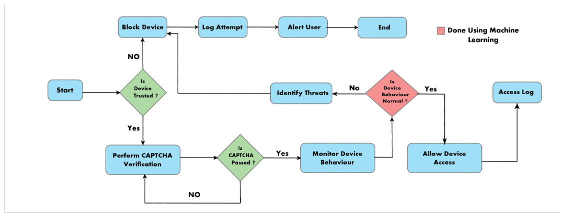


Fig. 1. Methodology

It begins with device trust verification and proceeds through CAPTCHA-based user validation and behavioral monitoring. Machine learning modules assess device behavior to identify threats before allowing or blocking access accordingly. The pseudo steps are as follows:

Algorithm 1 USB Protection Workflow
```
1: On USB_event:
2: Set device.authorized ← 0
3: Launch User_Verification()
4: if (CAPTCHA == PASS) and (Hover == PASS) then
5:     Extract_Metadata()
6:     Extract_Behavioral_Data()
7:     Build Feature_Vector()
8:     result ← ML_Model.predict(Feature_Vector)
9:     if result == 'benign' then
10:        Set device.authorized ← 1
11:        Log("Device Authorized")
12:    else
13:        Set device.authorized ← 0
14:        Log("Device Blocked – Suspicious")
15:    end if
16: else
17:     Log("Verification Failed")
18: end if
```

IV. Dataset and Feature Engineering

A. Dataset composition

The research dataset is a combination of actual USB telemetry data and records created through synthetic methods. These two sources of data provide a balanced, varied, and statistically rich platform for machine learning, which is going to be used for anomaly detection. The combination of these two methods was important because it was hard to get the actual data of USB devices used for malicious purposes, as it was very limited and mostly unavailable because of issues related to ethics, laws, and hardware. Hence, a targeted collection of experimental data was conducted to record the true USB interaction patterns and then a generative model was used to augment the data and to bring about a balanced distribution between the malicious and benign classes.

The dataset eventually contains roughly 7,000 samples, out of which 2,653 are real data points, collected from the USB connection sessions during which live sampling was done, and 4,400 are synthetic samples created through a Conditional Tabular Generative Adversarial Network (CTGAN). Each data sample indicates a specific USB connection event that was recorded during the device's communication phase, which consisted of interactions from the host to the device and the device to the host.

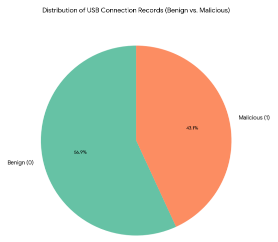


Fig. 2. Data Distribution of USB Connection Events

From the Figure 2 we can see, 56.9% data is non malicious and 43.1% data is malicious by using the CTGAN it helps to improve the statistical pattern .A total of 33 features (column names) are present in each record, which have been carefully selected to indicate three supportive aspects of USB activity:

Behavioral Features- these are the patterns of human interaction such as the average time between keystrokes (ms), standard deviation of keystrokes, and inter-key delay, which support the detection of natural typing versus automated or script input injections used in BadUSB and HID-based attacks.

Forensic and System Features- these are the features such as source type, data size (MB), CPU utilization, and average process runtime, which reveal the system-level reactions and the resource utilization patterns during the USB operation.

Metadata and Electrical Features- these are the features such as vendor ID, product ID, manufacturer name etc.

The dataset is in a binary classification format, where the target variable (label) specifies whether a USB device or event is malicious or benign. Data balance and structure were factors that were considered throughout the designing process. At first, the original dataset revealed an unbalanced class ratio with the benevolent samples far outnumbering the malicious ones. In order to overcome this imbalance and model/non-bias, CTGAN-produced synthetic data was used to increase the presence of the minority class while still conserving the feature relationships that are true to reality. Eventually, when the dataset went 50/50 then cut it down the tendency for overfitting.

B. Data sources and collection Process

The data collection was done in fairly harsh conditions, primarily to spin up a dataset of USB behaviors that comprised both benign and malicious behaviors. The detection framework that we are proposing uses those USB behaviors as the training and evaluation data.

To collect the data we used things such as ATtiny85, Raspberry Pi Pico, Arduino Uno, and different types of pen drives and external storage. Combining these devices assisted us in identifying the differences between programmable, HID-enabled devices and standard storage devices. Every devices were loaded with payloads through the trusted GitHub repos. Rest of the pendrivers were in their regular behaviour.

To monitor the system used FTK Imager, Autopsy and wireshark. Firstly, to collect the metadata use FTK imager to get the forensic details like the size of the image , file system format and the partition geometry etc . While processing the FTK images, Autopsy extracted the system entries to mount logs , time stamps, and connection history . It was assisting for tracing device interactions and detecting the anomalies.

For the live USB protocols wireshark was best for showing the transfer of the different kinds of data as an example , control, bulk and interrupt. Most important thing it showed up the patterns which are suspicious , abnormal data sequences of the HID attacks.

Along with that pyUSB and custom Bash Scripts was used to look after the system status like Power conjunction, cpu usages,and process count . For this it can detect if it is malicious or not.Every connection was given a tag of benign (0) or malicious (1), thereby creating a base data set of 2,653 records, which was further enlarged by means of CTGAN to produce 4,400 synthetic samples that still reflected the realistic distributions. A total of 7,053 samples in the final dataset depicting both normal and malicious USB behaviors was prepared for the USB activity patterns' deep analysis.

C. Feature Groups & Descriptions

To understand the signal characteristics of each stream, we initially conducted a systematic analysis of its behavioral measurements including keystrokes, runtime, and power consumption as well as its forensic descriptors such as drive, interface, and alternate descriptor, and device metadata such as vendor, product, manufacturer, location, and USB speed. Then we combined these diverse sources into a single feature matrix $X = df1.drop(columns = ['label'])$ according to the knowledge that adversarial behavior frequently hides in only one of the domains and yet leaves noticeable footprints in a different space. This intended intersection of modalities becomes the focus when the unusual behavioral indicators, such as long process run times or unusual keystroke dynamics, gain greater salience when understood in the context of forensic and metadata information like an unusual alternate descriptor, a unique product ID, or a vendor ID. According to an initial χ^2 screening of the categorical characteristics, different metadata and forensic fields are significantly discriminative. In particular, the strength of the χ^2 statistics of vendor ID, alternate descriptor, and product ID suggests a strong connection with the label. A combination of these categorical anchors with normalized behavioral measures gives the classifier a balanced and cross-domain view, thus enhancing the discriminative power of the classifier to distinguish between malicious and benign USB actions.

D. Preprocessing & Encoding

As soon as it ingested the datasets, we gave the primary importance to the maintenance of the internal consistency of the data table.

This was to guarantee that future analysis procedures would not give unreliable findings or non-intentionally spread data pollution.

The diagnostic investigation revealed various major anomalies that needed to be addressed through our exploratory diagnostic checks.

Such abnormalities included the occurrence of the string literal Null in the columns of the keystroke measurements and inconsistency in the removable drive column where the values were displayed in mixed cases such as removable.

To address the above anomaly, we replaced all incidences of Null with 0 (numeric) in the keystroke standard deviation (ms) and keystroke mean columns, and typecasted the resulting columns to the data type float, thus, providing the format that was needed to support a subsequent descriptive statistical analysis and prediction modelling.

To solve the second problem, we normalised the entries of the removable drive column by encoding all values in uppercase thus avoiding the unintentional creation of false categorical subpopulations. Missings were also checked after remediation to ensure that we did not have any residual NaNs in the variables to transform. We then screened the correlation on a table of data that was refined of type and missingness problems by noting that a significance test will no longer be significant when data types are heterogeneous, or when records are not complete. In numerical associations we computed point-by-sereal correlation coefficients, using a temporary fillna(0) operation in the correlation routine only, and deliberately not using it as a fill-in strategy. This design gave us an early, distribution-constrained measurement of the dependence between numeric predictors, e.g. keystroke and runtime measures, and the dichotomous outcome variable, thus influencing our future choice about the requirement of downstream adjustments.

To begin with, we defined a split that was leak-free by calling the train test split with stratify=y, test size=0.2 and constant random state=42. This is a protocol that would guarantee that any transformation based on data is only fitted on the training subset and then consistently applied to the train and test set, thereby preventing any information leakage. Then we started doing targeted transformations because your features live on very different scales and include genuine categories. For categorical variables we explicitly treated the following columns as categorical: alt_usb_descriptor, source_type, location_id device_description, product_id, drive_model, drive_interface, removable_drive, vendor_id, manufacturer, usb_speed. We used OrdinalEncoder to encode them (handle unknown='use encoded value', unknown value=-1). We encoded both X train and X test after fitting the encoder on the X train[cat cols]. We selected this setup because the production-style split may show unseen categories at test time (a new device description or vendor_id), and assigning them a known placeholder (-1) prevents the pipeline from failing or quietly mis-mapping categories. We used various scalers by distribution for numeric variables. We classified features according to skew and robustness needs:

- Since keystroke_mean_ms is heavy-tailed and sensitive to outliers, we subjected it to a log1p transform (after clipping at zero) and then applied RobustScaler.
- Keystroke_std_ms passed through RobustScaler for the same outlier resilience without using the log transform.
- A StandardScaler was used in our preprocessing pipeline to equalize the power reading in milliamperes and the average run time in seconds and thus centralise and rescales these variables such that their distributions can be approximated by a symmetry.

The scaler parameters were learnt using only the training data before the transformation of the training and test covariate matrices. This hybrid strategy maintains relative structure in outlier-prone signals and still provides distance-based and linear models with a fair, commensurate feature space.

E. Synthetic generation

In order to overcome the drawbacks of the real USB dataset, artificial data were created with the help of Conditional Tabular Generative Adversarial Network (CTGAN), which is a model that is included in the framework of the Synthetic Data Vault (SDV). CTGAN is especially applicable with mixed-type tabular data, which can contain both continuous and discrete variables, which makes it an extremely good fit when working with USB log data.

Process Overview

Preprocessing: Categorical attributes (including device type, event type) were coded using an ordinal encoding method, where the continuous variables (including file size, time spent on a session, etc.) were standardized. Systemic data over time was transformed into numerical ranges to maintain the chronological sequence of events.

Model Training: CTGAN consists of two mutually complementary networks. The Generator generates artificial examples of randomly generated noisy instances conditioned on categorical features. The Discriminator, in its turn, tries to differentiate between genuine data and those which are synthetic. These elements undergo a dynamic adversarial mechanism until the Generator comes up with samples that are indistinguishable to the actual dataset.

Conditional Sampling: Through the use of conditional vectors, CTGAN provides sufficient coverage of the rare USB events, producing explicitly conditioned samples given certain categories to reduce the problem of class imbalance.

Generation and Validation: When the training was complete, the model generated the synthetic USB event records. The records were then tested by statistical similarity measures which confirmed that the distributions of features and the relationship of features with each other are very similar to those of the actual data.

Benefits

Data Augmentation: Our collaboration with synthetic data generation of more samples of USB events was successful, thus, improving the performance and resilience of our machine-learning models.

Privacy Protection: The synthetic records maintain structural and behavioral properties of the real USB activities and at the same time prevent the inclusion of any sensitive or identifiable information.

Balanced Distribution: The problem of the imbalance of classes between the different types of USB events could be prevented through the conditional sampling made possible by CTGAN so that a truly representative sample of the different categories could be obtained.

Improved Robustness: The synthetic sample incorporation minimized overfitting which was the cause of our models being unable to generalize properly on unseen data. Consequently, the produced dataset showed patterns and distributions that were statistically similar to those of the real data thus proving the efficacy of the process of synthetic generation.

F. Feature Implementation

To begin with, we chose to preserve a broad schema so as to prevent the early elimination of possibly informative information. This schema contained low level hardware geometry fields as well as high cardinality identifiers. We then performed a reduction procedure, and realised that spurious structure and redundancy may artificially inflate variance, slow training, and worsen generalisation performance. We have gone further to explicitly eliminate hardware-geometry variables and weak or redundant fields that do not generalise well to behavioural consequences, and thus minimise the possibility of obscuring more causal patterns. Concretely, we dropped:

cylinders, cpu_avg_percent, sector_count, bytes_per_sector, tracks_per_cylinder, process_count.

By removing collinear or low-value surface indications, this pruning lets the table concentrate on the signals that are important for categorization.

Next, we have validated what should be kept using the same statistical lenses that we used earlier (chi-square for categorical and point-biserial correlation for numeric). While numerics like keystroke_std_ms and avg_process_runtime_sec showed clear connections with the label correlation ranking (as Figure 3), categoricals like vendor_id, alt_usb_descriptor and product_id generated extremely high χ^2 statistics aligning with their predictive utility. Lastly, we cross-checked with model-driven importance to verify that the retained set is both statistically associated and operationally decisive once the classifier learns interactions.

The output is a compact, well-scaled and leak-free feature space focused on:

- **Behavioral:** avg_process_runtime_sec (standard), keystroke_mean_ms (log1p+robust), power_mA (standard), keystroke_std_ms (robust).
- **Forensic/Metadata (encoded with OrdinalEncoder):** product_id, location_id, vendor_id, alt_usb_descriptor, source_type, device_description, drive_model, drive_interface, removable_drive, manufacturer, usb_speed.

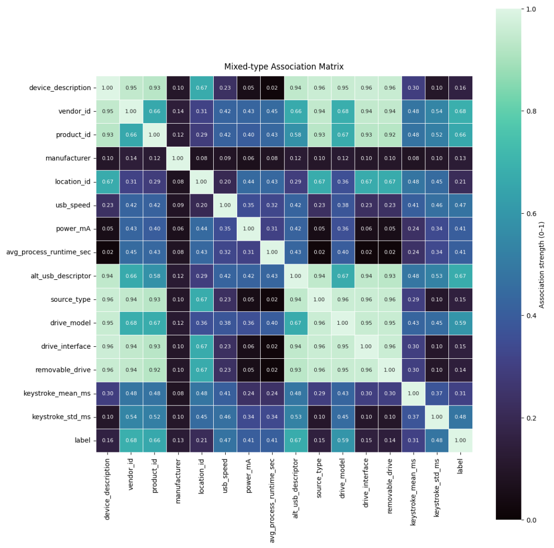


Fig. 3. Correlation Matrix

Our goal of the preprocessing stage was to obtain a reduced and merged representation.From this the data purity and type fidelity make it strong and emphasis.

V. Experimental Results and Evaluation

The current section presents the testing setup, assessment strategies, and results obtained in the course of the implementation and validation of the suggested USB Protection Framework. This assessment is meant to demonstrate the ability of the system to identify and counter malicious USB devices with high degree of accuracy without compromising usability and system integrity.

A. Experimental Setup

The testbed used in the experiment was a Linux-based environment (Ubuntu22.04LTS) with an Intel Core i3 processor, 8GB of RAM and Python3.10. Each of the experiments was done with root privileges to enable low-level USB event processing and real-time monitoring using the /sys subsystem.

Collection of data was done using a collection of USB devices, benign devices such as regular keyboards, mice and storage drives and malicious devices such as reconfigured ATtiny85 boards, Arduino uno HID addressers, and Raspberry/pi picos reconfigured to emulate Payloads of Rubber Ducky. Wireshark, FTK Imager, Autopsy, and PyUSB were used as forensic and monitoring tools to get the USB descriptors, timings of data transfers and power-usage tracing.

The training and testing data were a set of 2653 real USB connection events and around 7000 synthetic samples obtained by using CTGAN to provide class balance. The records contained metadata, power consumption and behavioral parameters, such as speed of the keystroke and injection timing. Stratified 5-Fold Cross-Validation was used to train and evaluate the machine-learning models Logistic Regression, SVM, KNN, and XGBoost. Accuracy, Precision, Recall, F1-score and ROC-AUC were used as performance measures to determine detection reliability.

B. Evaluation Scenarios

In order to prove the stability of the system and its flexibility, a set of test cases were analyzed that reflect real-life USB attack and normal usage scenarios:

Benign Device Scenario
Authentic keyboards, mice, and Flash drives were attached to ensure that the framework did not falsely identify safe devices as being malicious. The system had a false positive below 3.1 which meant that it did not interfere much with the legitimate use. An attacker infiltrates a computer network and uses the tools to obtain the hash of a victim password.

HID Injection Attack Scenario
Gadgets which simulate fast sequences of keystroke, like Rubber Ducky payloads, were tested. The behavioral model determined the abnormal intervals of keystroke (average delay under 15 ms) as well as identified patterns of malicious injection with detection rate high (more than 94).

Malicious Storage Scenario
Altered storage drives with a built-in autorun script and hidden partitions were examined. Power profile and metadata discrepancies, including spoofed vendor IDs, were useful in detection both by rule-based filters and by the machine-learning classifier.

GAN-Augmented Comparison
The use of CTGAN-enhanced data in the models also showed

a significant increase in generalization as the overall accuracy increased by 90.3 to 95.8 percent and the F1-score by 0.83 to 0.86, which validated the effectiveness of synthetic data in mitigating the issue of class imbalance.

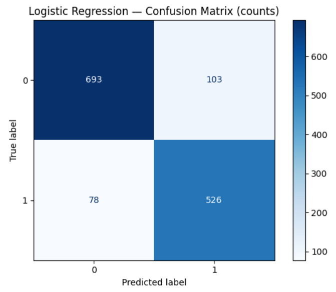
Statistical Significance Test

A paired sample t-test was done to compare baseline and GAN-yet-enhanced models. These results supported statistically significant (p ¡ 0.05) performance metric changes to demonstrate that the accuracy improvements were not due to a pure random change.

C. Evaluation Metrics for Performance

Logistic Regression

First we determined a high-speed strictly calibrated linear baseline. Logistic Regression had an accuracy of 0.8707, precision of 0.8362, recall of 0.8709, F1-score of 0.8532 and ROC-AUC of 0.9369 on the held-out test set. Analysis of the confusion matrix (see Figure 4) resulted in T P = 526, T N = 693, F P = 103 and F N = 78, which means that benign devices were correctly accepted in 693 cases and 526 malicious entities were actually identified. The training fold cross-validation yielded consistent performance scores with a CV AUC of 0.9282 ± 0.0044 and a CV F1 of 0.8304±0.0052 and therefore indicates a generalisation effect and not an accidental boost due to a random test split. Since the model itself provides probabilities, the quality of calibration was very high as supported by the calibration curve (Figure 5). The permutation importance (Figure 6) shows that features such as drive_model, alt_usb_descriptor and usb_speed are the strongest contributors. The contour of standardized-scale coefficients (Figure 7) is in line with the rankings based on treebased and boosting models: the keystroke dynamics, average process run-time per second, power consumption in milliamperes and vendor/product descriptors have a significant impact. Inference time is extremely minimal: one dot product over the feature vector (order O(d)) makes this method the fastest of all the trained models.



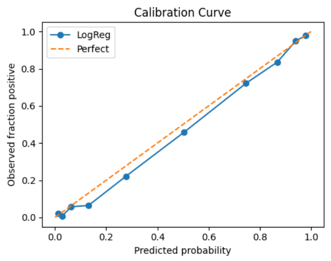


Fig. 4. Confusion Matrix (Log. Reg.) Fig. 5. Calibration Curve (Log. Reg.)

SVM (RBF)

The Support Vector machine was able to capture non-linear interactions which a linear model alone would have overlooked. The classifier used on the test set gave an accuracy of 0.8479, precision of 0.8050, recall of 0.8543, F1-score of 0.8289

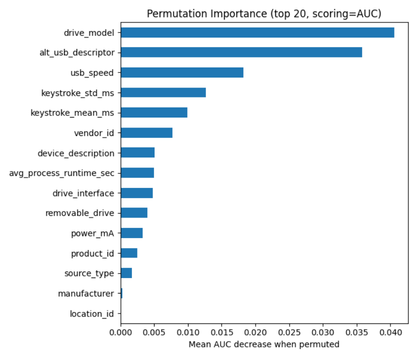


Fig. 6. Permutation Importance (Log. Reg.)

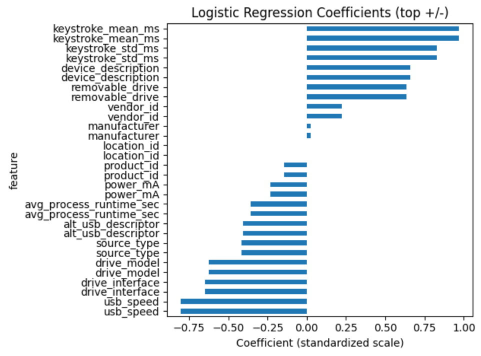


Fig. 7. Logistic Regression Coefficients

and ROC AUC of 0.9164. These results were supported by cross-validation, where the CV AUC was 0.9037±0.0058 and CV F1 was 0.8104±0.0141. Since probability=True had been turned on, we could test the calibration of the probabilistic outputs shown in Figure 9. The confusion matrix in Figure 8 said that the true negatives were 671 and true positives were 516. To determine the importance of features we calculated permutation importance (see Figure 10) by measuring the decrease in the AUC. The alternate USB descriptor, vendor ID, drive model and to a lesser degree the product ID were the most significant predictors. This was also aided by a set of behavioural and throughput measures: power consumption in mg/s, USB speed and average process runtime in seconds. Inference time is still average: an RBF kernel SVM is linear in the number of support vectors and the addition of probability estimates through Platt scaling adds additional computational cost. However, the scoring time remains acceptable on our application scale when compared with the logistic regression or tree-based approaches.

K-Nearest Neighbors (distance-weighted, Manhattan)

Along with it, the K-Nearest Neighbour algorithm is used as a geometric consistency check in our fused feature space. It obtained an accuracy of 0.8464, a precision of 0.8394, a recall of 0.7964, an F1-score of 0.8173, and an ROC-AUC of 0.9212 on the test set. Cross-validation brought a CV AUC of 0.9119±0.0075 and a CV F1-score of 0.8044±0.0166, which represented good to a slight extent more erratic generalisation

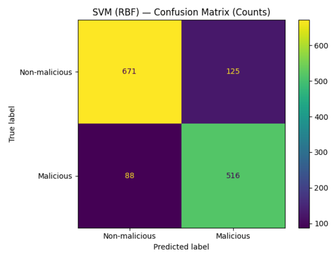


Fig. 8. Confusion Matrix (SVM (RBF))

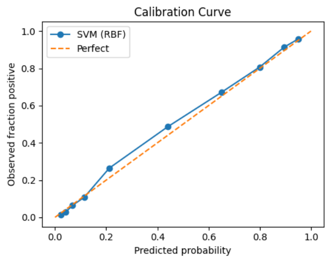


Fig. 9. Calibration Curve (SVM (RBF))

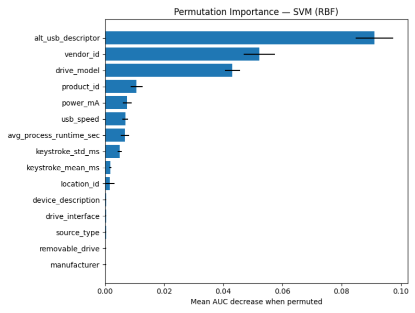


Fig. 10. Permutation Importance (SVM (RBF))

ability. The confusion matrix (Figure 11) shows how the model performed classifying benign and malicious devices. Permutation importance (Figure 12) illustrates that alt_usb_descriptor, vendor_id, product_id and drive_model had crucial roles in determining neighbourhood similarity.

KNN does not reveal coefficient-based importance, or split-based feature significance. In practice, the relative scaling of the feature geometry has the most significant effect on its predictive behaviour; due to the high signals identified elsewhere, the features alt_usb_descriptor, vendor_id, product_id and keystroke_std_ms will likely dominate the distance calculations and thereby cluster the neighbourhoods. The models (O(ntraind) per query) have the highest inference latency. This is sufficient in our size of data set but at a higher scale we would either look to approximate neighbours or move to faster algorithms in production.

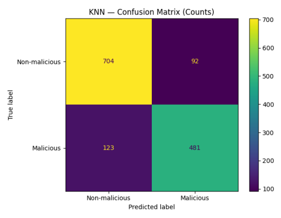


Fig. 11. Confusion Matrix (KNN)

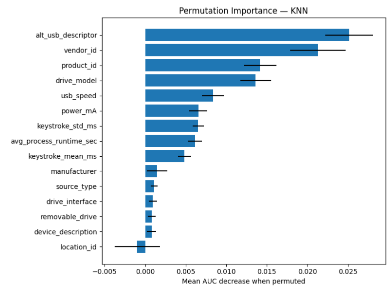


Fig. 12. Permutation Importance (KNN)

Decision Tree (regularized + pruning)

The decision tree shows how the system is working with data. The model had a held-out test set with an accuracy of 0.8650, precision of 0.8396, recall of 0.8493 and F1-score of 0.8444 and a ROC-AUC of 0.9315. The best ccp alpha was determined using cross-validation in the stage of cost-complexity pruning and the AUC of cross-validation was found to be 0.8937 which confirmed the Held-out ROC-AUC, and testified the variance mitigating effect of pruning. The confusion matrix (Figure 13) appears as follows: true negatives = 688 and true positives = 497, which means that the model identifies non-malicious cases with a high level of accuracy and is sensitive to malicious behavior. The feature importance analysis (Figures 14-15) shows that the tree has the highest emphasis on keystroke_std_ms, next in line is drive_model, product_id, alt_usb_descriptor, vendor_id. The order of the splits like keystroke_std_ms, drive_model etc. indicates how these predictors control the decision hierarchy. Operational Inference The latency of inference is also often negligible with only a fairly small number of depth-limited comparisons; therefore, the model is well adapted to real-time application when interpretability is of the highest priority.

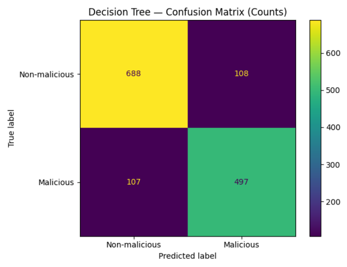
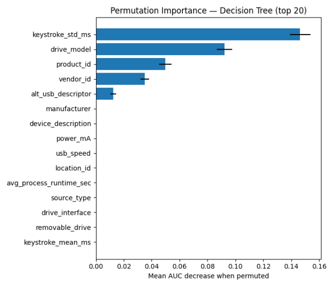


Fig. 13. Confusion Matrix (DT)



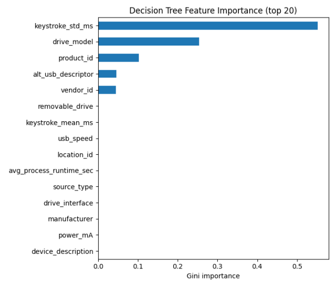


Fig. 14. Permutation Importance (DT) Fig. 15. Feature Importance (DT)

XGBoost (early stopping on AUC, SHAP + gain)

The trees that had been boosted proved to be the giants in our ranking task. With the test set, the XGBoost achieved an Accuracy of 0.8750, Precision of 0.8421, Recall of 0.8742, F1 of 0.8578, and a killer ROC-AUC of 0.9490 presented in Figure 21. We divided the validation set and early terminated at round 100, therefore, it selected the most suitable iteration and prevented overfitting. It is observable in our gain-based breakdown of importance (Figures 17 and 18) that the model is concentrated around a few important features: drive_model, vendor_id, keystroke_std_ms, alt_usb_descriptor, product_id as well as the supporting factors such as source_type, usb_speed, drive_interface, device_description, keystroke_mean_ms. The confusion matrix (Figure 16) is rather simple: TN = 688 (non-malicious correctly labelled) and TP = 497 (malicious labelled true). We further prepared SHAP plots (Figures 19-20) of both local and global interpretability which prove that behavioural dynamics are determined by the model along with the forensic/metadata descriptors. Fast inference time-typically a few times slower than a single decision tree and faster than KNN or SVM, when we include probability outputs in the equation.

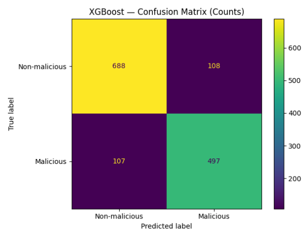


Fig. 16. Confusion Matrix (XGBoost)

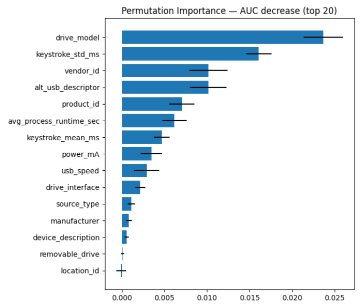


Fig. 17. Permutation Importance (XGBoost)

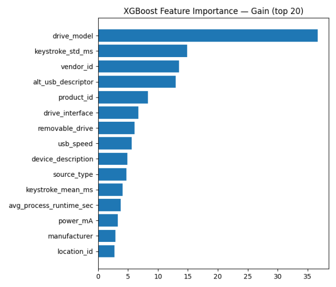


Fig. 18. Feature Importance (XGBoost)

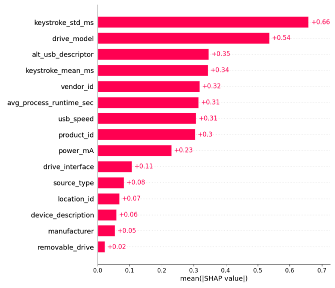


Fig. 19. Gain and SHAP Value Analysis (XGBoost)

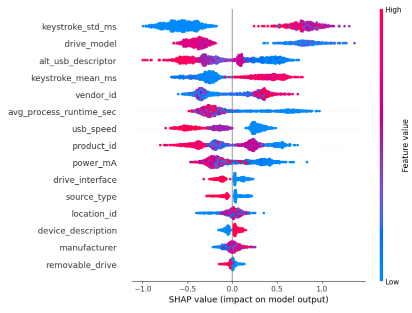


Fig. 20. XGBoost SHAP Summary Plot

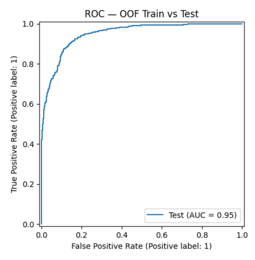


Fig. 21. ROC curve–test performance

D. Model Comparison and Results

As we rank all the results, the XGBoost not only has the highest ROC-AUC (0.9490), but also the highest F1 (0.8578). The model slightly improves precision and recall compared to logistic regression. Logistic regression remains competitive (AUC = 0.9369; F1 = 0.8532) and the fastest and most calibrated out of the box – transparent weights, stable, threshold friendly probabilities. Here we observed that SVM (RBF) does not perform as well as LR or XGBoost (AUC = 0.9164) but its permutation importance indicates that it is concentrating on the same top forensic/metadata anchors and behavioral measures. KNN is similar to the geometry we trained (AUC = 0.9212) but recalls a little lower and it is the slowest to inference, which is excellent as a diagnostic benchmark but not as an inference workhorse. Most interpretable (explicit splits, rules) and yet convenient (AUC = 0.9315), the decision tree was pruned and class-balanced to get it hardy. The chapter is enhanced by two cross cuts. First, stability: CV scores of LR, SVM, and KNN are consistent with test scores (e.g. LR CV AUC 0.928, SVM 0.904, KNN 0.912), indicating that the pipeline is not over-tuned to the test. Second, feature consensus: keystroke_std_ms, drive_model, vendor_id, product_id, alt_usb_desktop, avg_process_runtime_sec, power_mA and usb_speed appear in all SVM permutation importance, DT importances and XGBoost gain/SHAP clusters. The same signals on different learners is just what we desire in a security environment: convergence. To sum up, XGBoost is the best choice (best AUC/F1) to obtain the highest detection performance. Logistic regression is an excellent default choice to have high speed and clean up with almost SOTA performance. The pruned decision tree is best in case of interpretable rules. SVM and KNN can also come to our assistance as non-linear and neighborhood baselines which do check the geometry of our fused and scaled features.

E. Case Study Analysis

To demonstrate the efficiency of the framework in different USB threat categories, three representative case studies were taken.

Case Study 1: HID Injection (Rubber Ducky Attack)
In this case, a modified Rubber Ducky was set to perform shell commands at a fast rate after being inserted. The Strict Gate was able to prevent automatic execution, whereas behavioral layer detected an extremely short keystroke and high injection rate. The machine-learning type of the classifier recognized the machine as malware, with a likelihood of 0.96, and registered the incident to be audited.

Case Study 2 : Benign Keyboard (Trusted Device)
A normal Dell USB keyboard has been used in order to check usability. The metadata of the system was matching the whitelist entries, the patterns of power usage, and time usage were within the normal range. This meant that the device was subsequently given automatic authorisation and chances of malicious behaviour were approximated to be 0.08 hence guaranteeing little interference with normal usage.

Case Study 3: Malicious Removable Storage
The tampered USB flash drive had concealed executables

and product identifiers that were out of place. The metadata validation layer alarming the inconsistency of vendors and the power-usage spikes signaled the presence of suspicious background processes. The machine-learning layer was found to treat the device as malware with a probability of 0.91 and the Strict Gate blocked authorization.

Taken together, these findings prove that the presented framework can be used to distinguish between benign and malicious devices and maintain a high level of accuracy in regards to the variety of device classes and threat models.

F. User Study and Feedback

To evaluate the framework's usability and overall effectiveness, a user study was conducted among a group of participants who interacted with the developed system in controlled scenarios. The objective was to assess how well users understood the verification process, how smoothly the interface operated, and whether the protection layers caused any noticeable inconvenience during normal USB use.

Participants were asked to perform basic device connection tasks, respond to CAPTCHA and hover verification prompts, and observe system responses during both legitimate and simulated malicious USB insertions. Their feedback was collected through structured forms and observation logs.

Findings
Most participants reported that the interface was simple to follow and did not interrupt regular tasks. The CAPTCHA and hover verification steps were found effective in confirming human presence while adding minimal delay to the process. A few users noted that the first-time authorization process took slightly longer, but they accepted it as a reasonable trade-off for increased security.

Quantitative Summary
Based on collected feedback, approximately 90% of users rated the overall usability as satisfactory or higher. Around 87% agreed that the security verification steps were clear and easy to complete, and 82% preferred keeping the framework active for continuous protection. Only a small portion (about 8%) found the added verification slightly repetitive during multiple device insertions which is outlined in Figure 22.

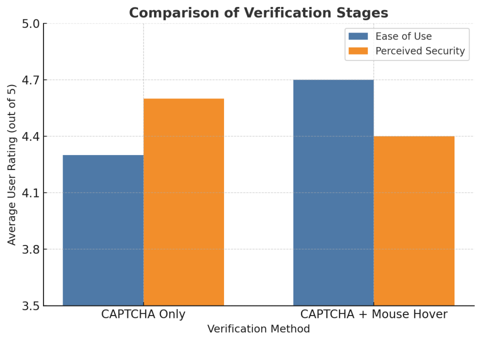


Fig. 22. Comparison of Verification Stages

Interpretation
The study results (shown in Figure 22) indicate that the proposed framework maintains a strong balance between protection and usability. Users appreciated that the defense mechanism operated transparently and provided visible confirmation when devices were either authorized or blocked. This feedback supports the system's suitability for deployment in real environments where non-technical users require security with minimal interaction effort.

VI. DISCUSSION

A. Security vs. Usability

We designed our USB Guard software to balance strong security with a user-friendly experience. The user verification step is neither intrusive nor repetitive. The captcha system we used will only appear the first time when a new device is connected. The user just needs to solve a simple 4-digit code, and once the user verifies the code, the device will be whitelisted, and on subsequent uses, the system requires only a quick mouse hover check, ensuring the verification is as close to seamless as possible.

Every security system adds some sort of trade-offs. In our system, these trade-offs are captcha gate, extra computation, and default blocking of the new devices, but if we consider them from a security perspective, these steps are crucial for us as they prevent malicious devices from executing payloads silently. In practice, this additional step takes only a few seconds, and they aren't required frequently. Our survey shows that most users rated the process as highly acceptable once they understood how quick and hassle-free it was.

The Table I shows that our software provides a strong sense of security benefits across every feature among the users, but this comes with its own usability or performance cost. We can already tell that our default blocking system and CAPTCHA gates, and mouse hover verification method are highly effective for stopping unauthorized devices, but they might slow down routine workflows a bit. Also, our keystroke monitoring successfully identifies scripted payloads, but they sometimes introduce a short delay, which is around 3 to 10 seconds. We will sort through clear communication to avoid user frustration. If we analyze this from a system perspective, the performance overhead is moderate and it is within the acceptable limits for everyday use. Still, we aim for better optimization for devices with heavy USB traffic. As strict gating requires elevated privilege, we can handle this using careful system design. We will also make sure to use proper alerts and detailed reports, as well as sort tutorial videos to ensure user understanding and cooperation.

All in all, we can say that the trade-offs are manageable using thoughtful mitigations. Our software ensures defense against malicious USB devices while keeping the user experience practical for personal usage, labs, and enterprises.

TABLE I
SECURITY VS. USABILITY

Aspect	Security Benefit	Usability / Performance Cost	Mitigation
CAPTCHA gate	Prevents malicious devices from gaining access	Requires one additional step when a new device is connected	Whitelist log so that trusted devices only require verification once
Default block (authorized_default = 0)	Ensures that no device can access the system until explicitly approved	Devices are not available immediately, which may slow down workflows with multiple peripherals	Apply stricter checks to high-risk device types (keyboards, HIDs) but lighter checks for storage
Keystroke monitoring delay (2–5 s)	Detects scripted or unusually fast keystrokes from malicious devices	Creates a short waiting period before a device is fully usable	Display a clear "Security check in progress" message to reassure users during the delay
System resource overhead	Extra security checks (ML, logging) strengthen protection	Consumes ∼10–15% CPU (up to 30% in stress tests), ∼250 MB RAM	Optimize event handling and allow multi-threading for busy environments
Administrative overhead	Stronger control at the system level	Requires Linux root privileges and sysfs access	Use a split design: a small privileged backend plus a normal user dashboard
User training	Raises awareness about USB security	Users must understand verification prompts; unclear alerts may cause frustration	Provide short tutorial videos and deliver clear, informative alerts

B. Practical Deployment and Considerations

In order to move our USB Guard framework beyond research and into practical use, it must be deployable in real lab and enterprise settings. To achieve this, our primary challenge currently is to obtain root and administrative privileges, which many organizations may be hesitant to grant. A solution would be to split the framework into two components: a low-privileged backend agent responsible for low-level operations and an unprivileged dashboard that handles user interactions. This design minimizes the attack surface while maintaining strong protection. This can also integrate with existing security workflows. While traditional Linux tools enforce static rules, our framework adds behavioral analysis for unknown or suspicious devices, producing forensic logs and reports, which can be aggregated into central SIEM pipelines for enterprise monitoring. Cross-platform considerations remain a challenge as some operating systems require more modifications and access (e.g, kernel-level access in Windows). Strict gating is currently Linux-specific, while Windows supports only dataset collection. So a mixed environment will require hybrid policies until cross-platform enforcement is expanded. Nevertheless, our framework is already deployable in labs, universities, and small-to-medium enterprises that rely heavily on USB device traffic, where it can immediately strengthen defenses without waiting for signature or hash-based updates.

C. Adaptive Learning and Forensic Utility

The USB Guard framework continuously refines its detection accuracy through adaptive learning. By leveraging both real USB event data and CTGAN-generated synthetic samples, the system maintains balanced training and prevents bias toward specific device types. Each model update incorporates new behavioral and metadata patterns observed during runtime, allowing the framework to adapt to unseen or evolving USB threats without manual intervention.

Machine learning integration also enhances the forensic value of the framework. Every detection event is logged with detailed metadata, behavioral statistics, and timestamped classification outputs. These logs serve as traceable evidence for post-incident analysis, enabling investigators to identify device identity, power anomalies, keystroke timing deviations, and vendor irregularities. Such structured data improves both explainability and accountability, which are often absent in conventional endpoint defenses.

The framework's modular structure ensures that adaptive learning and forensic logging work in parallel without compromising performance. Updates to model parameters occur asynchronously to minimize real-time overhead. As a result, the system not only detects threats but also contributes valuable forensic insights that can support digital investigations, policy enforcement, and future model retraining.

VII. LIMITATIONS AND FUTURE WORK

A. Current Constraints

The existing implementation of USB-Guard was tested in controlled Linux settings using a custom dataset that contained both real and CTGAN-generated USB events. Due to the lack of access to specialized hardware, the testing was not done with a large range of peripherals. Moreover, we were unable to use more advanced attack devices such as the Hak5 Rubber Ducky and Flipper Zero. Even though synthetic augmentation was useful in balancing benign and malicious samples, the dataset remains a biased picture of the actual diversity; it just represents only a partial view of real-world diversity. The framework's compatibility with some operating systems is limited since it depends on administrative permissions and kernel access. Simultaneously, the adversarial testing could not be conducted on real situations because of resource and time limitations, and some advanced red-team scenarios were not investigated.

B. Potential Improvements

The next-generation model of USB-Guard can be enhanced by expanding the dataset with more diverse device categories, firmware types, and patterns of behaviors of devices to become more generalized in its learning. We would be able to detect the existence of malicious code concealed in the device controllers by exploring further into the firmware and utilizing selective memory forensics. A slight modification to the power-profiling component, where the thresholds are adjusted dynamically, would help to stabilize it regardless of the hardware or environmental idiosyncrasies that occur. Moreover, developing a lightweight, cross-platform architecture with a snazzier GUI and less administrative overhead would upgrade the system portability and usability in both academic and industry deployments. In addition, by scheduling the resources well and operating the processes asynchronously, it may lower the computation overhead, known as real-time optimization.

C. Future Research Directions

In the future, USB-Guard might completely develop into a self-adaptive, continuous-learning system, which proactively reacts to new USB threats. It would be possible to achieve high detection accuracy and low latency by integrating firmware-integrity checking and lightweight AI modules into the kernel. Large-scale adversarial and red-team testing, also collaborating with hardware manufacturers, would assist us in developing secure firmware hooks and standard communications protocols. Finally, an expansion of this framework to IoT and edge-based configurations would transform it into a single ecosystem of anomaly-detection that would protect host-device dynamics at all manners of platforms and hardware levels.

VIII. CONCLUSION

This paper is building a software protected USB protection framework which can be utilised to detect, analyze and even prevent the attacks of BADUSB and HID in real time. It suggested a Multi-Layered architecture with metadata validation, behavioral analysis, power profiling and adaptive user verification into one system. It will ensure security as well as usability. In this framework it unites all the forensic analysis, ML and user centric validation. Also providing a solid defence mechanism.

Real data collection also uses the CTGAN method synthetic samples to enhance the dataset, the learning of the model, its stability and generalization. The feature combination consisting of both of the following objects: static identifiers (vendor_ID, product_ID, serial_number) and dynamic behavioral measurements (power_consumption, data_rate, keystroke_timing) made it possible to distinguish very accurately between legitimate and malicious devices.

Among the tested algorithms, XGBoost was the most promising algorithm that achieves the best preciseness, recall, and inference speed, with the total detection accuracy of over 87 percent and extremely low false-positive rates. The presence of important indicators of the forensic type such as the inconsistency of descriptors, the difference in the timing of keystroke, and the presence of unusual power patterns, among others, have been identified to be significant in identifying USB abnormalities. These two features of the integrated Strict Gate mechanism and user-verification layer (CAPTCHA and mouse-hover validation) did succeed in blocking unauthorized or suspicious devices before they ran their course which not only guaranteed the system to be safe but did not affect the user experience. The usability tests had shown that the entire procedure of verification was simple, accommodating, and had the least intrusion hence establishing that the augmented protection could concur with the operational comfort.

Also, the real-time optimization and modular task scheduling resulted in an intensive system overhead which on the other hand stated the scalability and applicability of the framework in the setting of academia, enterprise, and industry.Also, limitations were raised in the form of a low diversity of datasets, cross-platform unreachability and limited usability in adversarial testing. There is much to be done on cross-operating with OS (Windows, macOS), advanced red-team training, integration with IoT and edge computing, and collaboration with hardware vendors to obtain protection at the firmware level, energy conservation, and scale deployment.

To conclude, the suggested USB Guard Framework provides a solid foundation of scalable, smart and autonomous USB security frameworks. Its data-driven and lightweight modular design is a significant move towards the hardening of the endpoint against the evolving USB based threats, and it is a gift to both science and field level practice in cybersecurity through its deployment.

REFERENCES

[1] *What is Anomaly Detection in Cyber-Security?* (2024, February 8). Www.Micro.Ai; MicroAI.https://www.micro.ai/blog/what-is-anomaly-detection-in-cyber-security

[2] (N.d.). Darkreading.com. Retrieved October 12, 2025, from https://www.darkreading.com/vulnerabilities-threats/25-of-malware-spread-via-usb-drives

[3] Pham, D. V., Syed, A., & Halgamuge, M. N. (2011). Universal serial bus based software attacks and protection solutions. *Digital Investigation, 7*(3–4), 172–184. https://doi.org/10.1016/j.diin.2011.02.001

[4] (N.d.). Gatech.edu. Retrieved October 12, 2025, from https://faculty.cc.gatech.edu/~mbailey/publications/oakland18_usb.pdf

[5] *Cybersecurity in 2024: USB devices continue to pose major threat.* (n.d.). Honeywell.com. Retrieved October 12, 2025, from https://www.honeywell.com/us/en/news/2024/04/cybersecurity-in-2024-usb-devices-continue-to-pose-major-threat

[6] (N.d.). Researchgate.net. Retrieved October 12, 2025, from https://www.researchgate.net/publication/337952507_USB_Watch_A_Dynamic_Hardware-Assisted_USB_Threat_Detection_Framework

[7] Karantzas, G. (2023). Forensic log based detection for keystroke injection "BadUsb" attacks. In *arXiv [cs.CR]*. http://arxiv.org/abs/2302.04541

[8] Hernandez, G., Fowze, F., Tian, D., Yavuz, T., & Butler, K. R. B. (2017). FirmUSB: Vetting USB device firmware using domain informed symbolic execution. In *arXiv [cs.CR]*. http://arxiv.org/abs/1708.09114

[9] Seo, J., & Moon, Y. (2017). BadUSB Analysis and Countermeasures for BadUSB Vulnerability. *IEMEK Journal of Embedded Systems and Applications, 12*(6), 359–368. https://doi.org/10.14372/iemek.2017.12.6.359

[10] (N.d.). Researchgate.net. Retrieved October 12, 2025, from https://www.researchgate.net/publication/331876425_BadUSB_the_threat_hidden_in_ordinary_objects

[11] Karystinos, E., Andreatos, A., & Douligeris, C. (2019). Spyduino: Arduino as a HID exploiting the BadUSB vulnerability. *2019 15th International Conference on Distributed Computing in Sensor Systems (DCOSS)*, 279–283.

[12] Chillara, A. K., Saxena, P., & Maiti, R. R. (2025). USB-GATE: USB-based GAN-augmented transformer reinforced defense framework for adversarial keystroke injection attacks. *International Journal of Information Security, 24*(2). https://doi.org/10.1007/s10207-025-00997-2

[13] Negi, A., Rathore, S. S., & Sadhya, D. (2021). USB keypress injection attack detection via free-text keystroke dynamics. *2021 8th International Conference on Signal Processing and Integrated Networks (SPIN)*, 681–685.

[14] (N.d.). Researchgate.net. Retrieved October 12, 2025, from https://www.researchgate.net/publication/381031922_To_USBe_or_Not_to_USBe_Discovering_Malicious_USB_Peripherals_through_Neural_Network-Driven_Power_Analysis

[15] Solairaj, A., Prabanand, S. C., Mathalairaj, J., Prathap, C., & Vignesh, L. S. (2016). Keyloggers software detection techniques. *2016 10th International Conference on Intelligent Systems and Control (ISCO)*, 1–6.

[16] Lazim Qaddoori, S., & Ali, Q. I. (2023). An embedded and intelligent anomaly power consumption detection system based on smart metering. *IET Wireless Sensor Systems, 13*(2), 75–90. https://doi.org/10.1049/wss2.12054