

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

by

Shoeb Mahfuz

21301540

MD. Shadman Sakib Rahman

21301475

Most Sanjida Saklain

21301015

Naima Nawar Achol

23241088

Sadia Afrin

21301603

A thesis submitted to the Department of Computer Science and Engineering
in partial fulfillment of the requirements for the degree of
B.Sc. in Computer Science and Engineering

Department of Computer Science and Engineering
BRAC University
October 2025.

© 2025. BRAC University
All rights reserved.

Declaration

It is hereby declared that

1. The thesis submitted is my/our own original work while completing degree at BRAC University.
2. The thesis does not contain material previously published or written by a third party, except where this is appropriately cited through full and accurate referencing.
3. The thesis does not contain material which has been accepted, or submitted, for any other degree or diploma at a university or other institution.
4. We have acknowledged all main sources of help.

Student's Full Name & Signature:

Shoeb Mahfuz

21301540

MD. Shadman Sakib Rahman

21301475

Most Sanjida Saklain

21301015

Naima Nawar Achol

23241088

Sadia Afrin

21301603

Approval

The thesis/project titled “Enhancing USB Security: A Multi-Layered Framework for Detecting Vulnerabilities and Mitigating BadUSB Attacks” submitted by

1. Shoeb Mahfuz (21301540)
2. MD. Shadman Sakib Rahman (21301475)
3. Most Sanjida Saklain (21301015)
4. Naima Nawar Achol (23241088)
5. Sadia Afrin (21301603)

of Summer, 2025 has been accepted as satisfactory in partial fulfillment of the requirement for the degree of B.Sc. in Computer Science in October 10, 2025.

Examining Committee:

Supervisor:
(Member)

Muhammad Iqbal Hossain, PhD

Associate Professor
Department of Computer Science and Engineering
BRAC University

Thesis Coordinator:
(Member)

Md. Golam Rabiul Alam, PhD

Professor
Department of Computer Science and Engineering
BRAC University

Head of Department:
(Chair)

Sadia Hamid Kazi, PhD

Chairperson
Department of Computer Science and Engineering
BRAC University

Abstract

Universal Serial Bus (USB) devices are an inseparable part of modern computing and they are also a considerable cybersecurity threat. Malicious hardware like BadUSBs, Keyloggers and other peripherals that have been reprogrammed can pose as legitimate devices, execute commands that are not authorized and steal confidential data without the user's knowledge. This study 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, 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.

Keywords: BadUSB attack, Anomaly detection, Device fingerprinting, Forensic analysis, Adaptive learning

Table of Contents

Declaration	i
Approval	ii
Ethics Statement	iv
Abstract	iv
Dedication	v
Acknowledgment	v
Table of Contents	v
List of Figures	viii
List of Tables	x
Nomenclature	x
1 Introduction	1
1.1 Background	2
1.2 Rational of the Study or Motivation	3
1.2.1 Limitations of Existing Defense Systems	3
1.2.2 Relevance to Real-world Circumstances	3
1.2.3 Expected Impact of the Study	3
1.3 Problem Statement	4
1.4 Research Objectives	5
1.5 Methodology in Brief	6
1.6 Scopes and Challenges	7
1.6.1 Scopes of the Research	7
1.6.2 Challenges and Constraints	8
2 Literature Review	9
2.1 Preliminaries	9
2.2 Review of Existing Research	11
2.3 Comparative Analysis of Related Works	18
2.4 Research Gap Identification	19
2.5 Summary of Key Findings	20

3	Requirements, Impacts and Constraints	22
3.1	System Specifications and Technical Requirements	22
3.1.1	Software (Development & Runtime)	22
3.1.2	Hardware (Testbed & Deployment)	23
3.2	Societal impact of the USB Guard framework	24
3.2.1	Positive social and operational impacts	24
3.2.2	Potential negative externalities and social costs	25
3.3	Environmental Impact	25
3.4	Ethical Issues	26
3.4.1	Data minimization & anonymization	26
3.4.2	Consent and transparency	26
3.5	Standards Alignment with NIST / ISO / CIS Controls	26
3.5.1	Mapping table-contributions of USB Guard to standard func- tions	27
3.6	Project Management Plan	28
3.6.1	Project Schedule	28
3.6.2	Budget Plan	29
3.6.3	Resource Management	30
3.7	Risk identification and mitigation strategies	30
3.7.1	Risk scoring	30
3.7.2	Risk matrix	31
3.7.3	Mitigation mapping and operational controls	32
3.8	Economic Analysis	33
3.8.1	Cost Estimation	33
3.8.2	Benefit Estimation	33
3.8.3	Cost-Benefit Estimation	34
3.8.4	Interpretation	35
4	Proposed Methodology	36
4.1	Overall Research Methodology	36
4.2	Threat Model and Security Assumptions	38
4.2.1	Adversary Capabilities	38
4.2.2	Assumptions	38
4.2.3	Attack Tree Representation	39
4.3	Dataset Engineering and Preparation	40
4.3.1	Data Collection	40
4.3.2	Data Cleaning and Missing Value Handling	43
4.3.3	Data Transformation (Normalization, Encoding)	44
4.3.4	Data Integration	44
4.3.5	Data Reduction and Feature Selection	45
4.4	GAN-Based Dataset Augmentation	46
4.4.1	Motivation for Synthetic Data Generation	47
4.4.2	GAN Architectures Implemented	48
4.4.3	GAN Training Procedure and Hyperparameters	49
4.4.4	Validation of Synthetic Data Quality	52
4.4.5	Advantages and Limitations of GAN-Augmented Datasets	54
4.5	Model Design and Training Strategy	56

5	Result Analysis	59
5.1	Evaluation Metrics for Performance	59
5.2	Analysis of Design Solutions	70
5.3	Final Design Adjustments	71
5.4	Statistical Analysis	72
5.5	Comparison Between GAN Models	72
5.6	User Study Results	78
5.7	Discussion of Results and Implications	79
6	Software Implementation (USB Guard System)	81
6.1	Software Stack and Tools Used	81
6.2	System Architecture of USB Guard	83
6.3	Implementation of Core Modules	84
6.3.1	Strict Pre-CAPTCHA Gate	84
6.3.2	CAPTCHA Dialog for Human Verification	84
6.3.3	Keystroke Timing Monitor	85
6.3.4	Reporting and Alert System	86
6.3.5	Dashboard Design	88
7	Critical Analysis and Broader Implications	89
7.1	Security vs. Usability Trade-offs	89
7.2	Robustness Evaluation of USB Guard	90
7.2.1	Stress Testing under High USB Device Load	91
7.3	Comparison with Industry Tools	93
7.4	Deployment Considerations for Enterprises and Labs	95
7.5	Ethical and Privacy Implications of Behavioral Monitoring	95
7.6	Case Studies	96
8	Conclusion	98
8.1	Summary of Findings	98
8.2	Limitations of the Study	99
8.3	Future Work Recommendations	100
	Bibliography	104

List of Figures

1.1	Methodology	7
4.1	Attack Tree of USB Threats and Mapped Mitigations	39
4.2	Distribution of Benign and Malicious USB Records	41
4.3	Data Distribution of USB Connection Events	42
4.4	Correlation Matrix	46
4.5	Real Dataset Label Distribution	49
5.1	Confusion Matrix for Logistic Regression Model	60
5.2	Calibration Curve (LogReg)	60
5.3	Permutation Importance (LogReg)	60
5.4	Logistic Regression Coefficients (Top Positive and Negative Features)	61
5.5	Confusion Matrix for SVM (RBF) Model	62
5.6	Calibration Curve for SVM (RBF) Model	62
5.7	Permutation Importance for SVM (RBF) Model	62
5.8	KNN-Confusion Matrix	63
5.9	KNN Permutation Importance	63
5.10	Decision Tree-Confusion Matrix	64
5.11	Decision Tree-Permutation Importance	65
5.12	Decision Tree Feature Importance	65
5.13	XGBoost-Confusion Matrix	66
5.14	Permutation Importance (XGBoost)	67
5.15	XGBoost Feature Importance	67
5.16	Gain and SHAP Value Analysis	68
5.17	XGBoost SHAP Summary Plot for Feature Impact	68
5.18	ROC curve - test performance	69
5.19	Learning Curve-LogReg	74
5.20	Learning Curve-SVM (RBF)	74
5.21	Learning Curve-KNN	74
5.22	Learning Curve-Decision Tree	74
5.23	Learning Curve-XGBoost	74
5.24	Learning Curve-XGBoost	76
5.25	Learning Curve-Decision Tree	76
5.26	Learning Curve-KNN	76
5.27	Learning Curve-SVM (RBF)	76
5.28	Learning Curve-LogReg	76
5.29	Comparison of Verification Stages	79
6.1	System Architecture of USB Guard	83

6.2	CAPTCHA-Based Human Verification Interface in USB Guard	85
6.3	USB Guard Alert	87
6.4	Log Entry Example	87
6.5	Administrative Interface for USB Device Management	88
7.1	Stress Test Performance under USB Device Load	91
7.2	Attacker Effort Before vs After USB Guard	93
7.3	Comparative Feature Coverage between USB Guard and Existing USB Security Tools	94

List of Tables

2.1	Comparative Analysis of USB Security Defenses	18
3.1	Primary Runtime Components and Libraries	22
3.2	Host Machine Configurations	23
3.3	USB Device Models and Their Roles in Experimental Evaluation . . .	24
3.4	Mapping of USB Guard Security Capabilities	27
3.5	Project Schedule and Deliverables for USB Guard Framework	28
3.6	Estimated Budget Breakdown for USB Guard Project	29
3.7	Team Roles and Responsibilities in USB Guard Project	30
3.8	Identified Risk Matrix and Recommended Mitigation Strategies . . .	31
3.9	Economic Analysis: Cost–Benefit Evaluation over Three Years	34
4.1	Representative USB Feature Groups and Descriptions	42
4.2	Architecture of Conditional GAN (CTGAN) Components	50
4.3	Hyperparameters used for USB tabular data synthesis	52
4.4	Class distribution in the synthetic dataset	53
7.1	Security vs. Usability Trade-offs and Mitigations	89
7.2	Adversarial Strategies and USB Guard Countermeasures	92
7.3	Privacy Risks and Safeguards in USB Guard	96

Chapter 1

Introduction

The digital era depends on USB devices to establish data transfers while creating peripheral connections that link hardware systems together. The extensive use of USB devices creates multiple security weaknesses that cyber attackers exploit to execute their primary attacks. Research shows that USB-borne malware attacks grew substantially from 2019 to 2024, with a 42% rise that brought the total to 51% [15], [37]. Malicious USB devices can trick security systems by impersonating genuine peripherals to launch complex keylogging attacks, data theft operations, and firmware manipulation attacks [41].

Multiple high-profile cybersecurity organizations including, the Cybersecurity and Infrastructure Security Agency (CISA), have warned about increasing USB exploitation risks. USB devices used for malicious purposes can automatically type keys while delivering payloads and taking control of system functions without evading antivirus software or endpoint protection. Attackers deploy pre-programmed micro-controllers including, BadUSB, USB Rubber Ducky, and Arduino-based payload injectors to perform stealthy intrusions, which make detection more difficult [26].

Traditional USB security solutions primarily use software-based defenses to combat threats, but they do not protect against hardware vulnerabilities at the low level. The security gap requires multi-layered protection systems that integrate anomaly detection with real-time monitoring and power consumption profiling, behavioral analysis, and CAPTCHA-based authentication [27]. Our research creates an advanced detection framework that detects malicious USB activities through hardware and software analysis and user interaction monitoring to provide complete protection. Because the framework identifies new and unfamiliar threats without requiring continuous signature or hash updates. It analyzes behavioral and forensic features to detect zero-day USB attacks automatically instead of depending on predefined patterns. This makes the system adaptive and capable of handling new threats dynamically.

The proposed framework uses machine learning algorithms to analyze human behavioral patterns through keystroke dynamics, typing speed, and USB interaction latency for attack detection. The framework implements USB metadata validation and power consumption profiling alongside AI-driven anomaly detection models to stop unauthorized USB activity without compromising user convenience or system functionality [9]. The detection models are trained with both real and synthetic data using GAN-based augmentation to improve diversity and reduce class imbalance.

The growing reliance on USB peripherals throughout personal, corporate, and industrial environments makes USB communication security an essential requirement rather than an optional feature. The research actively identifies and mitigates malicious USB threats to strengthen cybersecurity protocols and minimize unauthorized access risks while offering a scalable, future-proof USB security model for individual users and enterprise-level infrastructure.

1.1 Background

Universal Serial Bus (USB) peripherals serve as fundamental components of contemporary computing systems because they facilitate effortless data transfer, device connectivity, and power delivery. The widespread adoption of USB peripherals throughout personal, professional, and industrial applications has simplified workflows and accelerated modern innovation. The widespread adoption of USB devices has created important security weaknesses that many users fail to recognize or often overlook.

The main security risk from USB peripherals emerges from their ability to be exploited by cyber attackers. Studies say that 25% of malware infections originate from USB devices which makes them one of the most exploited attack vectors globally [28]. Devices such as ATtiny85-based microcontrollers and USB Rubber Ducky tools function as legitimate peripheral imitators that bypass standard security protocols. The inherent trust in USB technology allows these devices to execute payloads such as keyloggers, data-exfiltration tools, and sophisticated threats. The growing number of hidden attack vectors requires stronger protective measures for system security.

The rapid development of USB technology in hardware and firmware capabilities has outpaced the security development rate. Alarming, global reports reveal a rising trend in USB-borne attacks with 9% of cybersecurity incidents linked to malicious USB devices [29]. Current security solutions primarily react to software vulnerabilities while ignoring threats that exist at hardware and physical levels. The lack of alignment between security development and hardware capabilities has resulted in substantial gaps within cybersecurity frameworks which expose systems to sophisticated threats that evade standard defense mechanisms.

Critical infrastructure sectors such as healthcare and finance together with industrial systems experience the most severe threats from malicious USB devices. Operations can break down while sensitive information leaks and financial and reputational damage becomes significant when a single USB device gets compromised. The destructive results of USB-borne attacks against industrial control systems and hospital networks demonstrate the necessity to develop proactive solutions [6].

Multiple security layers represent the necessary framework to reduce these threats. The combination of anomaly detection with real-time monitoring and adaptive authentication establishes a complete defense system against USB-based attacks. Early detection of malicious activity becomes possible through methods that combine device behavior anomaly detection with USB metadata validation and power consumption monitoring. These frameworks show the potential to transform USB security

standards through their ability to maintain user convenience while achieving cross-environment compatibility.

The research fills essential gaps in USB security through a comprehensive framework that combines advanced methods including device fingerprinting behavioral analysis and metadata validation. The proposed outcome creates an adaptable security standard for USB protection which defends personal users and essential systems against modern security threats.

1.2 Rational of the Study or Motivation

The rapid increase of USB peripherals in personal, professional, and organizational environments has introduced serious security threats. Though USB devices offer convenient data transfer, device connectivity, and power delivery, their inherent attributes turn them into dangerous weapons for malicious exploitation. This study is driven by the pressing need to address and mitigate critical vulnerabilities in cyber security caused by USB devices to ensure reliable and comfortable computing environments.

1.2.1 Limitations of Existing Defense Systems

Existing USB security systems mostly focus on software-level defenses and fail to address sophisticated hardware-level attacks. Malicious devices, such as USB Rubber Ducky, Arduino Pro Micro, ATtiny85 microcontrollers, and Hack5 tools, are capable of emulating legitimate peripherals to deploy covert payloads. These devices can go past traditional security systems and execute complex tasks like data exfiltration, keylogging, and injecting malicious commands [27]. Additionally, many current frameworks can analyze behavioral patterns or detect anomalies at the hardware and firmware levels, leaving critical vulnerabilities unaddressed and making USB devices a significant threat vector [27].

1.2.2 Relevance to Real-world Circumstances

USB-based threats are particularly severe in critical infrastructure sectors such as finance, healthcare, and manufacturing. A malicious USB device has the potential to expose sensitive information, disrupt essential operations, and cause substantial financial and reputational loss. Attacks on industrial management systems could cause expensive operational disturbance and malicious USB peripherals invading hospital networks could jeopardize patient data affecting the delivery of life-critical services. The above situations emphasize the importance of a dynamic and multi-layered defense strategy capable of correctly identifying and mitigating risks caused by USB peripherals.

1.2.3 Expected Impact of the Study

The goal of this research is to reduce the existing gaps in USB security systems by developing a strong framework that integrates advanced techniques, including:

- **Behavioral Analysis:** Anomaly studies of device behavior help detect malicious intentions.
- **CAPTCHA-Based Validation:** A user-friendly approach of verification to restrict unauthorized access [30].
- **Mouse-Hover Verification:** An effective verification method that observes real-time cursor movement to distinguish genuine user interactions from automated malicious activities [8], [19].
- **USB Metadata and Firmware Validation** Verify the metadata and firmware-level information of that USB device to measure its authenticity.
- **Power Consumption Profiling:** Monitoring of device power usage behavior to detect abnormal or suspicious activities caused by rogue devices.
- **Adaptive Threat Detection:** The system can identify new or zero-day USB attacks without relying on frequent signature or hash updates, ensuring continuous adaptability.

The proposed solution is designed to accomplish:

- Strong detection and prevention capabilities to identify and neutralize malicious USB devices in real time.
- Reduction of organizational disruptions and damages while maintaining user convenience and satisfaction.
- Protection of sensitive data and essential systems through multiple defensive layers across behavioral, firmware, and hardware levels.
- Software (does not require additional hardware), feasible for both personal and enterprise settings due to its scalable.

In essence, our research addresses cybersecurity challenges by developing a proactive and flexible software-driven defense framework that ensures reliable USB security in a variety of digital contexts.

1.3 Problem Statement

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.

1.4 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.

Detect Malicious USB Devices:

- A machine learning-based detection system would be used to examine and analyze human behavioral data (such as keystroke dynamics, typing latency, and mouse-hover patterns) along with USB metadata and power consumption profiles to identify risky USB devices.
- The system implements anomaly detection approaches that learn normal device behavior and identify irregularities in USB activity patterns in real time.

Mitigate USB Threats:

- Security protection functions through multiple layers combining human behavioral analysis with anomaly detection and threat identification to block USB-based attacks.
- Develop a threat mitigation system that stops recognized threats without interrupting regular user activities.

Enhance Usability and Accessibility:

- Users need access to a framework interface which enables simple system integration across multiple operational environments and devices.
- The users do not need to wait because the security is kept at a high level with the use of adaptive systems that connect CAPTCHA validation with device fingerprinting efforts.

Leverage Machine Learning for Anomaly Detection:

- Machine learning models need training and deployment to learn human-device interaction patterns before they can detect suspicious activity signals.
- Threat monitoring activities on collected insights support better detection mechanism development and threat adaptation.

Protect Data and Critical Systems:

- Protect sensitive data and maintain operational system integrity through protective measures in healthcare facilities and financial organizations and industrial production facilities.
- The security system should block only malicious USB devices yet allow legitimate ones without creating unnecessary blockages to protect users from false alerts.

At the end, the study provides a scalable, proactive defense mechanism that can be seamlessly deployed into personal or enterprise settings - protecting systems against a growing number of USB-based cyber attacks.

1.5 Methodology in Brief

To find out the vulnerabilities of USB peripherals and upgrade their security through advanced detection and prevention techniques, this study adopts a multi-faceted approach [21]. Firstly to detect lackings in existing security measures at both hardware and software levels, this study proposes an in-depth analysis of USB-related risks, such as rogue peripherals and malicious device behaviors. Next, A dynamic and multi-layered security framework is developed containing various user authentication methods including behavioral analysis, CAPTCHA verification, USB metadata validation, power consumption profiling, and device fingerprinting. This framework aims to conduct dynamic identification and prevention of malicious USB activities. To accurately achieve that, we created datasets collecting human behavioral patterns such as keystroke dynamics and timings for both legitimate and malicious devices to validate and support the training and testing of our security framework. Then, The detection framework is developed in the implementation phase combining machine learning models and rule-based algorithms, incorporating features like keystroke latency analysis, anomaly detection, and power consumption profiling. The framework will undergo precise and continuous testing of multiple USB peripherals using known malicious tools to validate its detection accuracy, response times, and false positive rates across simulated and real-world environments. The framework evaluation process focuses on threat detection capabilities as well as usability and scalability improvements promoting legitimate USB functionality and user convenience relative to existing solutions. Our research documents (shown the work-flow in Figure 1.1) all of our findings alongside implementation details and limitations while we concentrate on framework optimization through testing feedback and newly discovered constraints.

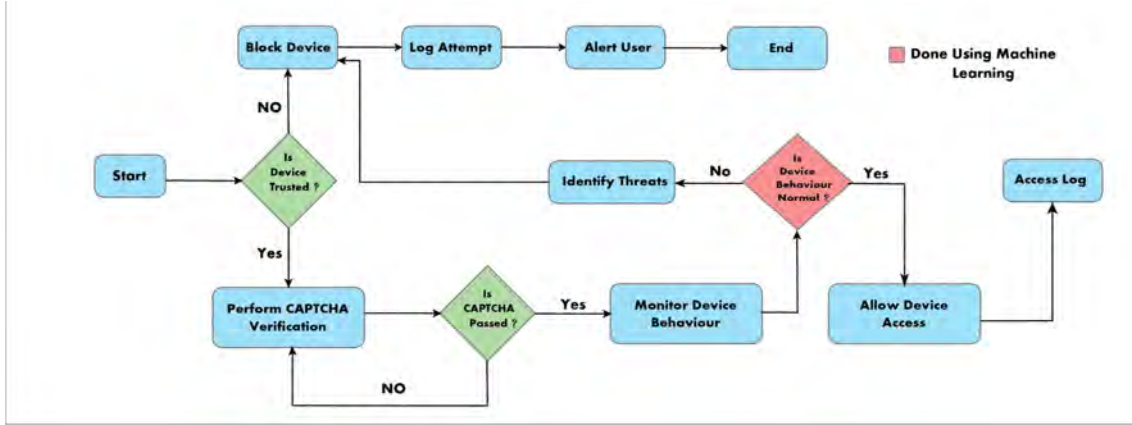


Figure 1.1: Methodology

1.6 Scopes and Challenges

1.6.1 Scopes of the Research

Proactive Threat Detection

In recent times, the identification of malicious USB devices has advanced detection techniques that are being developed, such as anomaly detection, behavioral features, and USB metadata and firmware validation.

Layered Security Measures

Deploying adaptive security elements that rely on CAPTCHA authentication, mouse-hover detection, and device fingerprinting to block unwanted device access attempts while maintaining smooth usability.

Integration with Diverse Environments

Design the framework to operate well with personal computers as well as enterprise networks and critical infrastructure sectors that cover healthcare, finance, and manufacturing industries.

Adaptive and Data-Driven Learning

Applying GAN-based data augmentation and optimized machine-learning models to enhance detection accuracy, robustness, and generalization against evolving attack vectors.

Real-Time Monitoring and Response

Develop real-time threat identification technology that promptly responds to dangerous occurrences while maintaining all authorized USB peripheral functionality.

Advancing Cybersecurity Practices

Keep contributing to developing USB security standards together with best practice recommendations that identify new attack approaches while maintaining user-focused designs.

1.6.2 Challenges and Constraints

Data Collection and Diversity

This represents significant challenges for collecting comprehensive human behavioral data, like keystroke dynamics, typing patterns, and interaction timings. Capturing a variety of datasets that are required from users with typing styles and habits while simulating real-time scenarios to include accurate and malicious activities for precise testing and validation.

Balancing Security and Usability

Users need security measures that combine ease of collaboration with intuitive design. The system must prevent the generation of pointless alert notifications.

Scalability and Resource Efficiency

The framework design needs to prove its ability to scale and use resources efficiently when operating between different platforms including enterprise systems and IoT devices without creating excessive operational or computational demands.

Adapting to Emerging Threats

The framework must maintain adaptability while remaining resistant to new attack techniques that attackers develop against USB vulnerabilities. The fight against emerging security threats requires continuous development and system updates as the fundamental method for protection.

Sensor and Power Profiling Accuracy

Due to hardware noise or controller variation, measuring USB power signatures reliably across different devices can be inconsistent.

Firmware Access Limitations

Some devices restrict access to firmware-level information; hence, it may require privileged system access.

GAN-Based Data Reliability

Though generating synthetic data helps balance datasets, but it is quite challenging to ensure realistic attack patterns and prevent bias in the trained models.

CAPTCHA and Mouse-Hover Usability

Must be carefully implemented to avoid interfering with normal user activities.

Real-Time Processing Overhead

Integrating multiple defense layers increases processing load. It must be optimized to sustain system responsiveness.

Chapter 2

Literature Review

2.1 Preliminaries

This section uses a brief outline of the essential technical underpinnings, conceptual frameworks, and critical terms that the research on the subject of USB security and anomaly detection are anchored on. It combines core concepts of hardware, software and user behavior to give the background that is necessary to the proposed study.

USB devices now serve as essential devices in the modern computing system, but since they can be used in such a wide range of applications and there is a level of trust in the system that the devices are trusted to be a target in advanced cyber-attacks [39]. Traditional security measures based on software level do not tend to identify low-level or behavioral threats which may be caused by reprogrammed or fake devices. To cope with this problem, the given ideas formulate the scope of this study.

BadUSB and HID Injection Attacks

BadUSB is merely a hack in which an individual reformats the firmware of a device to make it bad without it appearing otherwise [5], [31]. The most famous one is when USBs masquerade as keyboards or network cards and begin launching commands or malware. The commonest is the HID injection attack, in which the compromised USB is virtually turned into a keyboard by typing at a blistering pace and bypassing the normal antivirus and endpoint security [9], [13].

Behavioral and Keystroke-Based Anomaly Analysis

Keystroke dynamics are used to examine the individual typing behaviour such as key-hold time, inter-key delay and timing variance to detect whether a human or bot is typing [9]. Thus, when we are testing these bad USB devices, in particular the ones which do HID injections, observing that these devices have these very regular and somewhat artificial timing traces. Pulling these keystroke features into anomaly-detection models will allow us to intercept non-human interactions and correct accuracy on verification in the nick of time [11].

USB Metadata and Firmware Validation

All USB devices contain metadata such as vendor and product ID, serial numbers and driver information that can be forged by attackers [1]. Authentication of such data and verification of the integrity of the firmware provides good evidence to differentiate between the devices that are genuine and those that are spoofed or

compromised. On the other hand, firmware analysis tools, which can be thought of as symbolic execution frameworks, are able to reveal obscured logic or other unwarranted modifications at the microcontroller level [4].

Power Consumption Profiling

In a nutshell, the power-draw of USB devices may be quite informative concerning the possibility of some unsavory activity. When an unauthorized thing is booted, it will also tend to draw a strange current shape that is completely unlike the one a legitimate peripheral would draw. We can identify the behavior of the odd devices with a constant power profiling draw, trained in a machine-learning model, without having to look into the software signature, or install a few heavy tools [12], [27].

Device Fingerprinting and Real-Time Monitoring

With a combination of metadata, electrical signatures, and behavioral characteristics, we are able to develop a device fingerprint that actually identifies each peripheral in one session to the next. These fingerprints are utilized in real-time monitoring systems to track new connections, ensure that the correct individuals enter the premises and retaliate immediately when a red flag is raised. It does everything on the host itself, hence no additional gadgets are required [25], [32].

Generative Adversarial Networks (GANs) for Data Augmentation

The lack of labeled attack data remains a chronic problem in the research of USB security. The GAN-based synthetic data generation can be used to overcome this limitation as it generates realistic tabular records that imitate behavioral and power-profiling patterns of malicious and benign devices. These synthetic balanced datasets enhance generalization of the model, which helps anomaly detectors to detect both common and zero-day USB threats [22], [23].

Taxonomy of USB Threats

USB-borne attacks can be categorized across several interdependent layers:

Hardware Level: At the hardware level, attackers may embed hidden circuits or microcontrollers into what looks like an ordinary USB stick. Classic examples include devices like the USB Rubber Ducky or ATtiny85-based boards, which pretend to be keyboards and inject commands straight into the system. Since these attacks originate from the physical layer, they are often able to easily get past traditional software defenses, which makes them severely risky in sensitive systems [7], [16].

Firmware Level: BadUSB attacks take advantage of reprogrammable firmware in USB controllers. Attackers can repurpose a storage device into a Human Interface Device (HID) or network adapter, by rewriting the firmware. This would allow the covert execution of harmful payloads. Since antivirus tools generally scan only storage partitions, it leaves firmware modifications undetected. That’s why Firmware-Level threats are especially dangerous [5].

Behavioral Level: Behavioral threats occur when a USB device shows abnormal activity, such as registering keystrokes at inhuman speeds, abnormal process execution, or unusual timing patterns. Since HID injection attacks attempt to imitate user activity, they fall into the category of Behavioral-Level threats. To distinguish between human input and automated scripts, behavioral analysis and keystroke dynamics monitoring are necessary; they are the most sure-fire way to detect these

anomalies [3], [9].

Metadata Level: Every USB device comes with metadata like vendor ID, product ID, or serial number, that helps identify it. Attackers often falsify this information so a malicious device seems to be a trusted peripheral. Relying only on superficial identifiers makes detection weak, which is why newer approaches focus on deeper fingerprinting methods, sometimes even using power signatures to verify a device’s authenticity [1], [4].

Power Profiling Level: Power consumption patterns can greatly help in identifying malicious USB devices. If a device were to carry out additional unauthorized background operations, it would draw irregular amounts of current compared to the legitimate device. Power consumption profiling provides a supporting detection method by identifying these irregularities, especially when combined with other anomaly detection algorithms [12], [27].

Together, these areas present a multidimensional attack surface that requires a layered detection approach. A combination of behavioral, forensic, and physical-layer indicators will provide a good foundation of the USB Guard framework that we are offering in this paper.

2.2 Review of Existing Research

Now that we have a clearly planned strategy, we studied papers and articles that closely relate to our research from reputed sources such as IEEE, ResearchGate, and ScienceDirect to gain more insights. These studies address USB device security, hardware and firmware vulnerabilities, anomaly detection, and machine learning applications in cybersecurity. We learned more about the risks posed by USB devices, the evolving threat landscape, and advanced detection mechanisms. By reviewing these works, we have gained a comprehensive understanding of the challenges and opportunities in developing a robust and scalable USB security framework. Some of our readings are overviewed here:

To begin with, this study [31] evaluates the vulnerabilities within USB peripherals by tracing their evolution from USB 1.0 to USB-C. The paper addresses design flaws including lacking in authentication techniques and encryption in earlier USB versions, which make the systems vulnerable to unauthorized data access, firmware exploitation, and malware injection. It also addresses the threats associated with the availability of USB devices, emphasizing the pressing need for secure boot mechanisms, upgraded firmware protection strategies, and metadata validation to eliminate these vulnerabilities.

Furthermore, an article [37] demonstrated that there has been an increase in malware attacks transmitted through USB devices across industrial facilities. Research shows USB devices have become responsible for 51% of malware attacks in 2024 while their share has more than doubled in just five years from the original 9% in 2019. The rising trend demonstrates how industrial environments increasingly use USB devices to initiate cyberattacks. The report shows that content-based malware has become more prevalent because it exploits existing documents and scripting functions to represent 20% of detected malware. The analysis revealed that 13%

of blocked malware used Word, Excel, and PDF document formats to launch attacks. These findings demonstrate the urgency to deploy stronger cybersecurity standards by implementing strict USB device monitoring alongside state-of-the-art threat detection solutions together with public employee security training for keeping industrial facilities protected from USB-based dangers.

Moreover, the sectors of manufacturing and transportation along with healthcare and finance use removable media primarily through USB drives to execute essential operations such as software maintenance and firmware updates in air-gapped operational technology (OT) systems. This paper [36] addresses how USB devices have evolved into attractive targets for cybercriminals because they bypass security protocols and allow the entry of malware despite the 2023 Sogou malware attack which targeted multinational companies. The advancing complexity of attacks now includes malicious USB-based keystroke injection that allows attackers to break into secure networks to steal sensitive information. Removable media security proves difficult to manage because organizations lack standard security rules and incomplete visibility into file transfer operations which exposes numerous businesses to potential threats. The prevention of these security threats requires organizations to develop extensive scanning protocols together with Content Disarm and Reconstruction (CDR) methods and enforce stringent authorization controls and data encryption systems. A combination of continuous security audits alongside systematic employee training and multiple security measures establishes absolute protection against malware intrusion while reducing vulnerability to security incidents.

Ranney et al. (2025) [24] introduces a surprisingly clever technique called USB Snoo which is a side-channel attack, spying on user activity by taking advantage of how USB hubs manage traffic. The main motto of the method is that a seemingly harmless device like a regular USB device can be used to overload the USB connection with small bursts of data, causing delays. When these delays are precisely measured, it becomes possible to figure out what other devices are doing. For instance, when a user types on a keyboard, those keystrokes create specific timing patterns in data traffic which can be guessed by using machine learning techniques like Hidden Markov Models. Similarly, by watching the timing of network traffic from a USB Ethernet adapter, the system can monitor which websites the user is visiting- achieving the degree of accuracy over 83% using deep learning models. The paper also highlights that this type of attack does not require any special access or permissions and can work across USB 2.0, USB 3.x, and USB-C. But the research still has a deeper issue that the USB hubs can not isolate traffic between devices properly. It is just creating opportunities for these side-channel attacks. USB Snoo ultimately shows that USB traffic patterns themselves can be a goldmine for attackers since it can unintentionally leak sensitive information.

However, Koffi et al.[27] developed a machine learning system that uses power consumption analysis to identify malicious USB devices. USB security systems that operate at the software level fail to identify hardware-based attacks because they do not monitor hardware activities. The research suggests tracking USB power consumption irregularities as a method to detect harmful device behavior. The system utilizes real-time power readings which undergo Autoencoder feature extraction before LSTM and CNN models detect anomalies. The system achieves a near-perfect F1 score by detecting malicious devices through their unique power signatures which

result from hidden activities. Physical-layer security monitoring emerges as an effective non-intrusive framework that strengthens USB security frameworks while protecting against evolving USB-based attacks.

Though the enhancement of using USB peripherals has brought convenience to users, it has shown a new attack surface for hackers at the same time. The reason is that USB devices can act as dangerous tool, as they can execute harmful activity, be programmed to steal data, and inject harmful commands. To overcome this problem, researchers of this paper [9] developed a keypress attack detection system, which uses keystroke dynamics to distinguish between legitimate typing and malicious ones. This system is trained by recording a user's typing style with machine learning models and alerts when abnormal patterns are detected, meaning that one typing does not match the records of the system. Though the system performs very effectively within its measurements, there are some scope for improvements in it like- handling capital letters, special characters, and user mood variations. To wrap it up, with further improvements, this method can be a powerful cybersecurity tool to tackle USB keypress attacks, making our digital world a safer place.

Hernandez et al. (2017) [4] present a technical tool designed to inspect USB device firmware for hidden or potentially malicious functionalities. The main concern conveyed is- USB devices can sometimes pretend something that they are actually not like that. For example, some USB devices act as simple storage devices but they are actually capable of injecting keystrokes secretly on keyboards. To prevent this the researchers of this paper developed FirmUSB, a framework that uses a technique called symbolic execution. It allows the tool to explore different paths the program could take instead of running the firmware on the actual hardware. Symbolic execution also helps to uncover what the device might do in some certain conditions. The team customized their tools to handle that specific architecture and integrated them with well-known analysis engines like ANGR and FIE since many USB devices run on 8051 microcontrollers. The deep understanding of USB devices' structure makes the FirmUSB special- it knows what descriptors, endpoints, and class behaviors to look for, which makes its analysis more targeted and effective. Practically, FirmUSB is capable of detecting where seemingly harmless firmware was hiding dangerous capabilities. The findings basically point to the importance of firmware level validation when it comes to trustworthy device claims but in reality it is not. This work contributes a powerful tool for the security vetting of USB firmware, offering valuable insight into the opaque operations of embedded systems.

Additionally, the authors of this paper [2] have used a broad portion of USB technology for connectivity and data transport. Bad USB threat is a source for hackers, it allows malware to be embedded in the USB framework and remain undetected by security application systems. This research aims to know the vulnerabilities associated with this device and recommend security controls that are based on the NIST Risk management framework, and categorized into Technical, operational, and other controls. They have used the NIST SP 800-53 (rev 4) category in the technical, operational, and management controls. USB connections and logging were handled by technical controls. Critical information was created by management. Furthermore, they have used DriveCom and Duck encodes for the proper utilization of communication. The study validates its effectiveness against simulated BadUSB attacks, demonstrating its potential for enhancing system security.

Qaddoori and Ali [12] introduced a compact embedded anomaly detection solution for smart meters that employs ML algorithms to monitor strange power consumption patterns. This system operates on a Raspberry Pi 4 Model B unit to perform efficient anomaly detection in energy patterns with low computational resource requirements. The system applies a two-step machine learning approach with clustering algorithms (DBSCAN and Mean-Shift) identifying daily abnormalities and then using supervised classifiers (Decision Tree and Naïve Bayes) for detection plus unsupervised models (One-Class SVM, Isolation Forest, Elliptic Envelope) spotting periodic irregularities. The framework implements ongoing knowledge acquisition through data aggregator upgrades enabling model retraining using incoming meter measurements before updating smart meters. The system maintains effective usability through the combination of Real-time data exchange based on MQTT protocol alongside embedded security features and Bokeh web-based visualization capabilities. Experimental data reveals that DBSCAN delivers the best cluster results yet Decision Tree demonstrates perfect classification accuracy alongside One-Class SVM demonstrating the highest efficiency for periodic anomaly detection. The system is well-suited for IoT-based smart metering because of its low resource usage (29% CPU utilization with 2.31% memory use and 4.2W power draw). The proposed system delivers enhanced privacy protection through smart meter-based local operation along with faster detection speed and improved adjustment to changing power usage patterns while surpassing conventional cloud-based anomaly detection practices. The proposed framework decreases energy loss while boosting monitoring functions. It also provides scalable smart grid capabilities that will be improved through future work on cybersecurity developments and deep learning system integration.

The study [35] proposes a smart and practical approach to protect computers from threats carried through USB devices. In case of detecting irregular malware, traditional antivirus programs often fail in doing so; they are also very vulnerable to zero-day attacks. To address this issue, the researchers developed a system that checks the files against known malware databases with the help of VirusTotal API and also uses artificial intelligence to detect suspicious file behaviour. Both of these approaches combined allows the system to spot both previously seen and unseen threats. The program observes USB ports by scanning any recently connected devices and automatically takes action in real-time; it quarantines files with unusual behaviour and alerts the user via desktop pop-ups, emails or basically messages through any messaging service. The system is built with Python and machine learning libraries like Scikit-learn and TensorFlow. It is very lightweight and efficient, as it is able to scan a 16GB USB device in no more than ten seconds without straining the computer hardware. The program is also cross platform, working flawlessly on Windows, Linux and macOS, making it suitable for usage in various fields. The research emphasizes how artificial intelligence is capable of being used effectively to enhance cybersecurity in real-time, offering a necessary defense against modern USB-based threats.

Additionally, in this article [10], Fernandes and Lina explore how a small, custom-made device using an ATMEGA32U4 microcontroller and an ESP8266 WiFi module, can be turned into a wireless fake keyboard, sending keystrokes to a computer without any physical interaction. The point of this thesis is to show how attackers could use such a device to take control of a computer remotely, simply just by plug-

ging it into a USB port and transmitting commands over WiFi. The researchers successfully built and tested the device, showing how easily this kind of attack can happen. The main purpose of the study is to raise awareness about this security risk, not promoting hacking. Besides, it also encourages to strengthen the defenses by limiting USB access to both individuals and organizations to secure their devices more effectively.

On the other hand, Solairaj et al.(2016) [3] discusses how to detect software keyloggers- those sneaky programs can record everything you type, and can be detected before they cause harm. These keyloggers are designed to run silently and avoid detection by antivirus software. The researchers of this paper explore several smart detection strategies to detect the keyloggers. One approach looks for programs that hook into the system to watch you do. Another approach is called HoneyID uses fake input to trick the spyware and expose them. They also use a different technique called Spearman's Rank Correlation to detect bots by analyzing suspicious behaviors like- accessing files and sending keystrokes. A more advanced method they explore, called Dendritic Cell Algorithm, inspired by how the human immune system works, helps to identify abnormal behavior on a system. The paper even covers real life examples, such as- a case where an iPhone's sensors were used to capture keystrokes from a nearby keyboard. Finally, the author highlights mobile keyloggers as a growing threat and suggests using machine learning methods like Support Vector Machines for future detection since the study focuses on improving user safety and data protection.

Here, the paper [1] outlines the dangers of everyday USB devices by describing how attackers can reprogram USB devices to hide their identity. For instance, turning a flash drive into a fake keyboard that can automatically type malicious commands or hide malware in what seems like a safe storage drive. These kinds of attacks are tricky because computers typically trust USB devices the moment they are connected. The authors look at current ways to protect against these threats like blocking unknown USBs, isolating suspicious devices in secure environments, and tracking unusual behavior. But the current methods have their limits since the USB protocol is flexible and does not have the built-in security. The report ultimately asserts that we require more robust, sophisticated, and intelligent security systems capable of detecting and responding to anomalous or suspicious USB activity in real time to avert breaches of security.

Seo and Moon [5] made a detailed examination of the BadUSB vulnerability, pinpointing the proposed threats from USB devices capable of posing as imitation peripherals by substituting their firmware. The authors underlined that conventional antivirus software is unable to identify such attacks, as they do not scan the firmware level, where malicious code is present; rather, they normally scan USB devices' storage partitions. Their approach divides the USB firmware into read-write and read-only areas. And they suggest a two-phase solution to guard against it: i) hash-based verification of read-only descriptor fields, and ii) digital signatures on changeable firmware code. This can catch any unauthorized modification early, and only authorized firmware updates can be applied. They also had a secure update process for firmware, which provides extra protection. This work formed the foundation of hardware integrity-based USB security solutions by showing a change of paradigm in emphasis, toward low-level device activity could be the foundation for

an effective defense.

Blanchet [33] provided a thorough analysis regarding detailed exploration of the BadUSB attack, describing the covert dangers hidden in the firmware of seemingly harmless USB devices. Unlike traditional malware, BadUSB attacks bypass typical detection by exploiting the reprogrammable area of USB microcontrollers to masquerade as trusted peripherals such as keyboards or network cards. This permits surreptitious operation of malicious commands, traffic redirection, or data leaving without raising suspicion. Blanchet identifies the main security flaw as the general absence of firmware-level protection, specifically the lack of code signing and integrity checking. The work highlights the fundamental flaw in the current management of USB trust and applies to any USB-attached peripherals, not just USB drives. This work confirms the industry’s urgent need to adopt firmware authentication protocols to block such low-level, long-term attacks.

Karystinos et al. [7] proposed Spyduino, a hardware attack proof-of-concept that leverages an Arduino UNO board that was compromised to exploit the BadUSB vulnerability to emulate a Human Interface Device (HID). Unlike the conventional USB-based malware which requires payloads or storage, Spyduino runs a pre-programmed script by emulating keystrokes and can thus carry out malicious actions like switching off firewalls, stealing user information and uploading files to an FTP server without seeking any user consent. The novelty of Spyduino, however, is its stealthiness, or the fact that it is built into a standard USB keyboard and does not need special drivers, which makes it practically undetectable by a conventional antivirus. The authors emphasize that Spyduino can be employed to attack multiple operating systems, exploits passive FTP to avoid detection and leaves minimal artifact in system logs. Such countermeasures are mentioned as device control software, behavior monitoring software such as Duckhunt, and physical port blockers. The authors note that the measures usually do not apply to stealthy hardware-based attacks.

Next, a paper [14] indicates keyloggers as a serious cybersecurity threat that is capable of stealing and manipulating sensitive information like personal data and passwords. The situation is worsening as the keyloggers are getting more sophisticated and evading traditional antivirus detection methods. The paper studies numerous detection and prevention techniques such as the use of network traffic analysis, biometric authentication, antivirus, and antimalware software to tackle this matter. To protect sensitive data, many processes are introduced, including the use of advanced encryption, user education on social engineering, and conducting regular system updates. This study’s future work focuses on developing a stronger biometric system involving machine learning and artificial intelligence to enhance detection and prevention capabilities. The study also emphasizes putting effort into sharing the best practices and threat intelligence to overcome this challenge against keyloggers.

Similarly, this paper [18] explores serious security concerns involving stealing data and unauthorized software installations related to keyloggers. To begin with, effective monitoring tools are needed that can track user activities without being detected by any antivirus software. This paper talks about developing a software-based keylogger using Python, which records keystrokes and analyzes typing patterns to upgrade security measures. According to the study, a real-time keylogger was created that logs keystrokes and clipboard data while ensuring the recorded

information is safely stored and analyzed. Additionally, a feature selection method was implemented to improve the accuracy and efficiency of the detection process. For future work, the authors focus on exploring keylogger applications in broader contexts regarding behavioral analysis and risk assessments. That is why they also suggest upgrading keylogger's stealth capabilities to avoid detection by antivirus software aimed at improving internet users' security and privacy by an online keylogger detection framework.

This study [34] also explores the growing field of a type of malware named keyloggers that poses a significant threat to personal and professional data security as it stealthily records user keystrokes. It is challenging for users to protect themselves because keyloggers can operate undetected. This study shows a strategy involving the development of a stealthy software-based keylogger using Python to monitor the activities of a victim without informing them. This keylogger has various features including capturing keystrokes, clipboard monitoring, screen capture, and microphone activation for comprehensive data collection. This process spreads the malware through external storage devices or email keeping it hidden from antivirus software. The future work of this paper focuses on improving user awareness, upgrading detection methods, and developing stronger security measures to prevent these stealthy applications. This study also focuses on the need for ongoing research into the legitimate uses of keyloggers and ethical implications such as forensic investigations and parental monitoring to balance data security with privacy concerns.

On the other hand, Denney et al. [32] have used the USB-Watch for the hardware system to detect malicious USB events in recent times. In this function the USB devices and host computers the USB communication for detecting abnormal activities. It takes user input to analyze the threats through timing variations and data stream patterns. The system functions autonomously from the host operating system which results in both efficiency and lightweight performance. The accuracy of USB-Watch has been confirmed through testing which yielded a Receiver Operating Characteristic (ROC) score of 0.89 demonstrating its capability to differentiate between safe and dangerous USB activities. The research demonstrates that hardware-assisted tools represent a dependable defensive mechanism for protecting USB systems.

Finally, In this era, threats to security emerge from USB drives because they offer quick file storage alongside unrestricted device access. Attackers use USB drives to spread malware while infecting production devices and performing data theft by accessing computers physically. An attacker can recover information from a computer's memory even when the device is completely switched off. USB drives that are lost or stolen represent a major security threat because unsecured backed-up data remains vulnerable to unauthorized access. Protect your data from threats by never using unfamiliar USB drives alongside enabling encryption, passwords and separating personal and work storage, and turning off Autorun functionality while keeping both security software and systems fully updated. This document emphasizes the role of USB devices. This document [40] was published by the CISA and they suggested reducing the critical factor vulnerabilities associated with USB device usage.

2.3 Comparative Analysis of Related Works

To contextualize our proposed framework, we compare it with prior studies on USB security defenses. The following Table 2.1 highlights their core methods, strengths, and limitations.

Table 2.1: Comparative Analysis of USB Security Defenses

Paper (Year)	Core Method	ML Model(s) and Evaluation	Key Findings	Strengths	Limitations
USB-GATE (2025)[23]	Gateway-based defense; GAN adversarial training + transformer embeddings	Adversarial augmentation + transformers; robustness evaluation	Improved resilience to adversarially crafted keystroke injections	Strong adversarial ML design; resilient in keystroke domain	Gateway overhead; keystroke-only; no firmware/power
Forensic log-based detection (2023)[11]	Correlates OS logs (PnP events, drivers, processes) with keystroke anomalies	Anomaly detection on logs + keystroke features; latency histograms	Injected keystrokes show unnatural timing distributions vs. humans	No hardware required; leverages existing logs	Post-event forensic only; log-limited; no firmware/power
USB-Watch (2019)[32]	Inline hardware dongle monitors USB traffic; ML on HID anomalies	Decision Trees, Random Forest; ROC ≈ 0.89 on HID injections	Reliable HID detection; weaker on throttled attacks	OS-independent; tamper resistant; validated on real attacks	Requires per-port dongle; costly; HID-only; no firmware/power
To USB or Not to USB (2023)[27]	Power side-channel: fine-grained current traces classify devices	CNN and LSTM; $F_1 > 0.9$; CNN faster, LSTM better temporal capture	Power traces shown as effective behavioral fingerprint	Novel physical-layer perspective; strong accuracy in lab	Sensitive to hardware variation; ignores HID/firmware; lab-limited
USB Keypress Injection Detection (2021)[9]	Free-text keystroke dynamics (hold time, DD, UD latencies)	KNN, SVM, Random Forest, XGBoost; RF generally strongest	Free-text viable for continuous detection; avoids fixed passwords	Mature keystroke feature set; practical runtime; flexible input	Keystroke-only; no firmware/power; mimicry possible

According to the Table 2.1, it becomes clear that most prior approaches from the

existing research focus on limited aspects of USB security. Methods like keystroke patterns analysis, forensic log inspection, hardware dongles, or power-trace monitoring are addressed as only part of the problem in each study. Instead, we move beyond these isolated methods by combining multiple modalities of protection into a unified system. We fuse firmware data, power signals, and timing behavior with human-interaction patterns to capture a more complete view of device activity. Our main novelty of the work is that our approach operates directly on device behavior instead of relying on code-level inspection or external hardware, through a host-based design which is both practical and scalable. To further strengthen defense against automated keystroke injections, we also introduce CAPTCHA-based verification step and mouse-hover verification that helps to differentiate between legitimate and malicious activities. Additionally, our model significantly reduces false positives and improves reliability in real-time detection through the incorporation of all these signals. Unlike hardware-dependent methods, our framework achieves this entirely through software-based mechanisms that integrate forensic data, behavioral cues, and power profiling. Furthermore, traditional protection tools often depend on continuous signature or hash updates to detect or learn new threats, but our model can recognize unknown or zero-day USB attacks automatically by analyzing behavioral and forensic patterns without requiring regular updates.

To sum up, despite the significant contributions of earlier research, each study is still limited to a single defense layer. USB-GATE (2025) strengthens keystroke protection through adversarially robust ML; forensic log analysis (2023) outlines that host logs can reveal HID abuse, but only after the system has already been compromised; USB-Watch (2019) validates ML-based traffic monitoring, yet its reliance on specialized hardware limits practicality; To USB or Not to USB (2023) demonstrates the potential of power profiling but its effectiveness is hampered by hardware variability; USB Keypress Injection Detection (2021) confirms the utility of free-text keystroke dynamics, but its scope is confined to keystroke-level defenses. Combined as a whole, these pieces show discrete improvements that fall short of offering complete security. Our thesis addresses this shortcoming by integrating these perspectives into a multi-layer framework that emphasizes device behavior rather than code.

2.4 Research Gap Identification

Research on USB security and anomaly detection has advanced a great deal, but important limitations remain that reduce the effectiveness of current approaches against modern USB threats:

Limited Coverage of BadUSB and HID Injection Attacks

Most prior studies focus on detecting malware through signatures or endpoint protection. Firmware-level threats like BadUSB, and HID injection attacks where devices impersonate keyboards or mice, are less studied and still lack dedicated detection frameworks.

Insufficient Integration of Behavioral Features

Although anomaly detection has been explored, only a few studies integrate keystroke dynamics or other detailed behavioral indicators. Without these, systems may struggle to separate legitimate user activity from automated input generated by malicious

USB devices.

Lack of Multi-Layered Defense Frameworks

Existing approaches often rely on single detection techniques, such as metadata validation or anomaly detection. Except, attackers frequently combine multiple evasion methods. Therefore, single-layer defenses turn out to be inadequate. A more comprehensive framework that integrates multiple techniques such as behavioral analysis, device fingerprinting, metadata validation, and real-time monitoring is still missing from the avant-garde.

Data Scarcity and Class Imbalance Issues

Malicious USB devices are rarer than benign devices, so acquiring diverse and balanced datasets on them is quite difficult. USB threat detection research suffers from a lack of such datasets, as most available ones are skewed. This imbalance reduces the reliability and generalizability of machine learning models. Few studies have tried to solve this problem with methods such as GAN-based data augmentation for tabular data.

Usability and False Positive Concerns

Some proposed defenses disrupt normal work by blocking legitimate USB devices or by generating high false positive rates. As of now, research emphasizing usability and ensuring that security mechanisms run smoothly without interrupting users' workflow is still enormously limited.

Limited Focus on Critical Sectors

Only a small number of studies have evaluated USB defenses in these environments, though USB threats pose serious risks to various industries such as healthcare, finance, and industrial systems. Most work remains at a general or experimental level.

Identified Research Gap

From these findings, it is clear that a multi-layered, scalable USB defense framework is needed, that can:

- Detect and mitigate BadUSB and HID injection attacks in real time.
- Combine behavioral analysis (such as keystroke dynamics) with metadata validation and power profiling.
- Use GAN-based data augmentation to address data scarcity and class imbalance.
- Maintain high usability and low false positive rates, so that defenses can integrate smoothly into both personal and enterprise systems.

This gap forms the motivation for the proposed research, which seeks to build an adaptive, intelligent, and practical framework for protecting against the growing range of USB-borne attacks.

2.5 Summary of Key Findings

This study builds upon existing research and foundational studies which revealed important trends and challenges alongside methodologies for USB security. Below is a summary of the key findings:

Robust Multi-Layered Security is Crucial

The research demonstrates that USB security requires multiple detection methods

because a single approach fails to protect against all USB threats. The number of USB threats continues to evolve into sophisticated and complex forms each day. A multi-layered defense strategy delivers the most effective protection because it combines hardware checks with behavior analysis and software validation. The multilevel security model improves threat recognition accuracy while reducing improper alarms and detecting new emerging risks. Modern USB security solutions depend on these security measures to function as their essential foundation [13].

USB Devices as Vectors for Sophisticated Attacks

Malicious USB devices, like ATtiny85 microcontrollers and USB Rubber Ducky, can disguise themselves as legitimate peripherals while secretly running harmful programs in the system, which leads to threats. Since traditional security solutions focus on software-based threats, they fail to detect device-level vulnerabilities. Hence, a stronger security framework is highly required due to the limitations of traditional security measures. The security frameworks must incorporate hardware verification, firmware integrity checks, and behavior-based anomaly detection to prevent unauthorized USB activity and mitigate potential risks [25].

Evolving Patterns in USB Threats

Behavioral analysis techniques (such as keystroke patterns and power consumption anomalies) have proven the importance of sophisticated detection methods. The trained system implemented using machine learning by recording the USB activities can distinguish between malicious USB devices and legitimate ones. The study showed that this approach is proactive, which allows security systems to detect new attack vectors. It can also adapt to evolve threats, which makes it a powerful modern USB security solution [27].

Real-time detection Enhances System Integrity

Studies show that real-time systems are far more effective compared to periodic detection of malicious activities, as they greatly reduce the response time in taking initiatives, minimizing the damage by a significant amount. However, it is extremely important to ensure that the systems are well optimized and that they do not interrupt the actual USB operations [20].

Need for Scalable and Adaptive Frameworks

Pre-existing solutions are sometimes unscalable, and this limits their usefulness in IoT ecosystems, especially since USB devices are used in a wide range of environments. Thus, its scalable frameworks (that also have proper performance and accuracy) are essential for extensive adoption [38].

Balancing Security and Usability

The main lesson learned from this research shows that security protocols must not interfere with authorized USB operations. The proposed framework needs to achieve both strong protection measures and user-friendly design elements to gain successful adoption [17].

The review demonstrates that an effective solution requires a security framework that combines multiple adaptive layers and user-focused protection methods. The proposed research merges advanced techniques including real-time monitoring with behavioral analysis and scalable implementation to bridge existing solution gaps and establish new USB security benchmarks.

Chapter 3

Requirements, Impacts and Constraints

3.1 System Specifications and Technical Requirements

This project requires both a reproducible software environment and a controlled hardware testbed. The list below records the concrete software versions and hardware models we used during development and evaluation; keeping versions explicit supports reproducibility and helps reviewers reproduce experiments.

3.1.1 Software (Development & Runtime)

All components were developed using Python and commonly available open-source tools. The environment used for experiments is listed here; where appropriate, we add installation hints.

Primary runtime & libraries:

Table 3.1: Primary Runtime Components and Libraries

Component	Purpose in project	Version (used)	Install / Notes
Python	Primary implementation language for dataset builder, USB Guard	Python 3.10+ (recommend 3.10 or 3.11)	Use <code>virtualenv</code> or <code>venv</code>
pyusb	USB device enumeration & queries (pyusb + libusb backend)	pyusb 1.2.1	<code>pip install pyusb</code>
pyudev	udev event observer (Linux)	pyudev 0.24.0	<code>pip install pyudev</code>
evdev	Keystroke timing capture (Linux)	python-evdev 1.5.0	<code>pip install evdev</code>
pandas	Dataset manipulation & CSV I/O	pandas 2.1.0	<code>pip install pandas</code>
numpy	Numeric array operations	numpy 1.24.x	Required by ML stack
scikit-learn	ML pipeline, RandomForest	scikit-learn 1.2.2	<code>pip install scikit-learn</code>

Component	Purpose in project	Version (used)	Install / Notes
joblib	Model persistence	joblib 1.2.0	Used with sklearn pipeline
pywin32 / pynput	Keystroke capture (Windows dataset builder)	pywin32 304, pynput 1.7.6	Windows-only
Autopsy	Forensic analysis (parse FTK outputs for ground truth)	Autopsy 4.22.1	Used to generate FTK-style reports
FTK Imager	Forensic image creation (FTK text report)	FTK Imager 4.x (latest stable 4.x)	Export text reports for parser
Wireshark	Packet and USB traffic capture (optional)	Wireshark 4.x	Useful for USB-level troubleshooting
Tooling / Packaging	Logging, GUI, installers	tkinter (GUI), psutil, joblib	Install via pip/apt as needed

Summary of the basic software packages, libraries, and forensic tools that we used throughout the project are tabulated in Table 3.1. It contains the purpose and version of each component and how each can be installed to ensure that we can recreate the development and testing environment of the USB Protection Framework.

Notes on reproducibility:

- Reproducible experiments were run inside a dedicated Python virtual environment (`venv`) to control library versions. We include a `requirements.txt` in the artifact bundle listing the exact versions used during evaluation.
- On Linux, a working `libusb` implementation and proper permissions (or `sudo`) are required for `pyusb/pyudev` to enumerate and control devices.

3.1.2 Hardware (Testbed & Deployment)

The hardware list contains both the host machines used for development and representative USB devices used for benign and adversarial testing.

Host machines (example configs used in evaluation):

Table 3.2: Host Machine Configurations

Role	Model examples	Configuration
Development & Evaluation workstation (primary)	Intel® Core™ i7-9750H laptop or AMD Ryzen 7 5800X desktop	16–32 GB RAM, NVMe SSD, Ubuntu 22.04 LTS, USB 3.0 ports
Lightweight deployment target	Intel NUC or similar	8–16 GB RAM, SSD, Ubuntu Server
Embedded testbed (optional)	Raspberry Pi 4 Model B	4 GB RAM, USB ports (for small deployments/testing)

We used hardware configurations in the dev and test phases that are listed in Table 3.2 to track them. It identifies the key workstations, lightweight deployment

platforms and embedded testbeds to ensure that the evaluation is maintained to be similar across the hardware.

USB devices used in experiments (specific models):

Table 3.3: USB Device Models and Their Roles in Experimental Evaluation

Class	Device model (used)	Role in testing
Benign keyboard (reference)	Raspberry Pi Pico (RP2040) flashed as HID keyboard / Arduino Pro Micro (ATmega32U4)	Representative human keyboard input device
Malicious HID adversary	Hak5 Rubber Ducky (DuckyScript), Digispark (ATTiny/Arduino)	Emulated scripted keystroke injections
Benign removable storage	SanDisk Ultra USB 3.0 (consumer flash drive)	File transfer, forensic imaging
Malicious storage (payload)	Reprogrammed mass storage with autorun / staged payload	Payload injection tests
Forensic target (image capture)	External HDD / USB flash imaged with FTK	Source for FTK-style reports

Table 3.3 contains the names of the particular USB devices we used during the experiments and both benign and malicious. All types of devices were selected to provide the replication of realistic attack and normal conditions, which contributes to the behavioral and forensic analysis of the framework.

Peripherals & Measurement Tools:

- USB hub with per-port power measurement (optional) for power profiling experiments.
- USB protocol capture (Beagle USB 12/480/5000 or equivalent) when detailed bus traces are necessary (optional but recommended for advanced studies).

3.2 Societal impact of the USB Guard framework

This section frames the broader societal value and externalities of deploying a device-level defense that combines a strict gating policy, human verification, behavioral signals, and ML judgement.

3.2.1 Positive social and operational impacts

- Reduction of workplace infection vectors. USB devices remain a widely used human-carried vector for initial compromise (spear-phishing physical drop attacks, pre-programmed HID). A gate that prevents unspecified devices from acting until human verification and behavioral observation reduces successful initial footholds in small business, labs, and critical-infrastructure environments.
- Auditability and post-incident traceability. The reports generated (device metadata, FTK fields, keystroke timing metrics and model verdicts) create

an auditable trail that improves post-incident forensics and reduces time-to-containment.

- Lower entry barrier for defenders. The solution targets environments (university labs, research groups, small enterprises) that may lack complex EDR suites, giving them an evidence-based, open-source option.
- User education and security posture. The CAPTCHA/human-verification step serves a dual purpose of stopping automated attacks and increasing user awareness of device trust decisions.

3.2.2 Potential negative externalities and social costs

1. Usability friction: Frequent CAPTCHA prompts or false-positive blocks may frustrate users and drive them to disable protections, reducing long-term adoption. The user study in Chapter 6 quantifies this trade-off.
2. Privacy considerations: Behavioral features (keystroke timing statistics) and forensic metadata can be sensitive; careless storage or linkage with user identities can create privacy risks (see 3.3).
3. Deployment disparities: Facilities without administrative capacity (e.g., remote field sites) may be unable to operate the strict gating mode (root+udev), which could widen disparities in protection.

3.3 Environmental Impact

Our proposed framework has very little direct environmental impact since it is a software-based security system designed for increasing the safety of USB devices without necessarily involving the manufacturing of other hardware or requiring extensive deployment which may be demand intensive. Unlike hardware-driven security systems which involve in producing electronic waste and carbon emissions while manufacturing, this project prioritizes sustainability through utilizing existing computing infrastructure and digital innovation for implementation. Our framework operates conveniently on standard devices and requires no specialized components. As a result, it minimizes energy consumption and increases the functional lifespan of computers by reducing the risk of USB-caused threats that might lead to device servicing or replacement. The lightweight architecture and optimized algorithms of the framework make the use of power and computation overhead very low, which is compatible with eco-friendly software practices. Furthermore, since the system promotes secure and responsible digital behavior, it indirectly assists in the sustainability of information systems because data breaches and corresponding recovery costs will be minimized. To summarize, our system has a very minimal environmental impact as it does not produce any significant carbon emissions generated beyond routine computing operations which makes it an environment friendly cybersecurity solution.

3.4 Ethical Issues

The system captures behavioral signals (inter-keydown timings) and device forensic metadata (serial numbers, FTK image hashes). This section outlines ethical safeguards and data governance principles to minimize harm and ensure responsible research practice.

3.4.1 Data minimization & anonymization

- **Collect minimal aggregate statistics:** The implementation stores summary statistics (mean inter-keydown, standard deviation, event counts) rather than raw keystroke transcripts. This reduces the risk of leaking typed content.
- **Avoid storing identifying text:** Any logs that inadvertently contain typed content must be filtered and not retained. Serial numbers and device identifiers are stored only for incident traceability and should be hashed or access-restricted in long-term archives if they can identify individuals.

3.4.2 Consent and transparency

- **Deployment policy:** Organizations deploying USB Guard must notify users that device interactions may be subject to behavioral analysis and auditing. In workplace contexts, this should be included in acceptable-use policies.
- **Research consent:** For user studies (Chapter 6), participants must provide informed consent for collection of keystroke timing metrics and dashboard usability feedback. All questionnaire data must be anonymized prior to publication.
- **Retention windows:** For routine operations, keep detailed device reports (that include any potential PII) for a limited period (e.g., 30–90 days) unless part of an active investigation. Aggregate model telemetry (feature distributions, anonymized counters) may be retained longer for research.
- **Access control:** Restrict access to reports and raw datasets to authorized investigators. Use role-based access control and encrypt at-rest artifacts (e.g., AES-256 on archived report bundles).
- **Model details and artifacts:** While publishing synthetic datasets and model architectures aids reproducibility, detailed instructions that enable trivial replication of the strict-gate bypass should be avoided. Where necessary, redact or gate release of potent exploit recipes while releasing defensible details for academic review.

3.5 Standards Alignment with NIST / ISO / CIS Controls

To make the framework defensible in enterprise contexts, we align its functions to widely accepted security standards. This mapping shows how USB Guard contributes to standard control families and where supplementary organizational controls are needed.

High-level frameworks used: NIST Cybersecurity Framework (CSF: Identify, Protect, Detect, Respond, Recover) and ISO/IEC 27001 (information security management controls). We also reference NIST SP 800-53 control families for technical control labels.

3.5.1 Mapping table-contributions of USB Guard to standard functions

Table 3.4: Mapping of USB Guard Security Capabilities

Feature / Capability	NIST CSF category (function)	ISO/IEC 27001 clause / control area	Example NIST SP 800-53 control family
Pre-connection strict gating (authorized_default=0, udev enforcement)	Protect (PR): Access Control / Protective Technologies	A.9 Access Control; A.12 Operations security	AC (Access Control), SC (System & Communications Protection)
CAPTCHA human verification	Protect (PR) / Detect (DE) (prevention + detection of automated devices)	A.9, A.7 Security awareness	AC, AU (Audit & Accountability)
Keystroke timing behavioral monitoring (summary stats)	Detect (DE): Anomalous activity detection	A.12 Operations / A.18 Compliance	SI (System & Information Integrity), AU
ML classifier & decision logs	Detect (DE) / Respond (RS)	A.12 Logging & Monitoring	AU (Audit), IR (Incident Response)
Report generation & forensic artifacts (MD5/SHA1, FTK fields)	Respond (RS) / Recover (RC)	A.16 Incident management; A.10 Cryptography (hashes)	IR, CP (Contingency Planning)
Audit trails & model performance telemetry	Detect (DE) / Identify (ID)	A.12 / A.18	AU, SI

Table 3.4 fundamentally demonstrates how the security features of the USB Guard framework comply with the large cybersecurity standards. All capabilities are associated with the corresponding NIST Cybersecurity Framework functions, ISO/IEC 27001 control clauses, and NIST SP 800-53 control families, in other words, it is shown that the framework meets the rules and actually helps in the areas that are important, such as access control, incident response, and maintenance of the system in solid.

Notes:

- The mapping is intentionally high-level: adopting USB Guard in an enterprise also requires policy-level controls (acceptable use, change control, training) and integration with SIEM/IR playbooks to fully satisfy the listed standards.

- NIST SP 800-53 control families most relevant are AC, AU, IR, SI, and MP (Media Protection) for removable media handling.

3.6 Project Management Plan

We developed a project management plan to ensure cost-effective, efficient, and timely execution of the framework. Our plan includes budget allocation, timeline, and resource management strategy across the entire development phases. We tried to ensure standard software engineering practices for building our framework.

3.6.1 Project Schedule

To build and progress the framework, it took approximately 18 weeks, and the entire project took around 42 weeks. The details of each phase are mentioned below.

Table 3.5: Project Schedule and Deliverables for USB Guard Framework

Phase	Duration (weeks)	Main Activities	Deliverables
1. Project Initiation & Requirement Analysis	Week 1 to 6	Define project objectives, scope, and constraints; gather requirements; study USB vulnerabilities and existing defenses	Project charter & requirement specifications
2. Literature Review & Design Planning	Week 6 to 12	Review academic & industry research; finalize framework architecture and methodology	Design blueprint & methodology document
3. Dataset Collection & Preparation	Week 12 to 24	Collect real USB activity logs across different OS; use FTK Imager and Autopsy to retrieve data; perform cleaning, labeling, and normalization; test with basic ML models	Collected dataset of 2,653 records & baseline ML results
4. GAN-based Synthetic Data Generation	Week 24 to 26	Train the CTGAN model and validate 7,000 synthetic records for class balance and distribution fidelity	Augmented dataset
5. Model Development & Integration	Week 26 to 28	Train ML models (Logistic Regression, SVM, KNN, Decision Tree, XGBoost) and evaluate with cross-validation	Trained and validated classifier
6. Software Implementation (USB Guard)	Week 28 to 38	Develop multi-layer system using Python (PyUSB, PyUdev, Tkinter GUI); integrate CAPTCHA and fingerprinting modules	Working software prototype

Phase	Duration (weeks)	Main Activities	Deliverables
7. Testing & User Study	Week 38 to 40	Conduct performance evaluation, stress tests, and usability survey with participants	Evaluation results & survey report
8. Documentation & Final Report	Week 40 to 42	Prepare thesis, visual layouts, and finalize submission	Final thesis document

We have sketched the whole timeline as per Table 3.5 and subdivided it into phases based on the work to be carried out such as requirement gathering, data preparation, model training, code writing and final documentation. The timeline indicates that we will be moving step by step within the 42 weeks and ensure that nothing goes out of order.

3.6.2 Budget Plan

Our estimated budget for completing this project was approximately 30,500 BDT. The cost structure mentioned below emphasizes open source software and shared hardware resources.

Table 3.6: Estimated Budget Breakdown for USB Guard Project

Category	Items / Tools	Estimated Cost (BDT)	Remarks
Hardware Resources	Raspberry Pi Pico, USB Rubber Ducky, benign USB devices	22,000 BDT	Purchased for testing and dataset creation
Software & Licensing	FTK Imager, Autopsy, Wireshark (free version), Python libraries (open-source)	0 BDT	Open-source stack minimizes cost
Cloud / Compute Resources	Google Colab Pro for GAN training, and other subscriptions	5,000 BDT	Used for GPU-based CTGAN training, ML model training, software support, writing support, and research
Miscellaneous & Contingency	Printing, testing consumables, and others	3,500 BDT	Support & participant incentives
Total Estimated Budget		30,500 BDT	

The financial distribution of the USB Guard project has been estimated and presented in Table 3.6 in the form of hardware, software, and operational resources. I realised that the larger share of the budget was spent on the hardware buying, and we saved money on software by using open-source. The remaining was made up of cloud resources and some small contingencies that increased the overall estimate to an approximate of 30,500 BDT.

3.6.3 Resource Management

Human Resources:

Table 3.7: Team Roles and Responsibilities in USB Guard Project

Role	Responsibility	Assigned Member(s)
Project Manager / Coordinator	Schedule tracking, resource allocation, progress reporting, and problem resolution	Md. Shadman Sakib Rahman
Dataset Engineer	Dataset cleaning, data analysis, and synthetic data generation	Naima Nawar Achol
Machine Learning Lead	Feature engineering, model training, and evaluation	Most Sanjida Saklain
Software Developer	Python implementation, PyUSB integration, GUI development, FTK Imager analysis, and Autopsy logs	Shoeb Mahfuz
Testing & Documentation Officer	User study, survey data collection, and final report editing	Sadia Afrin

Technical Resources:

- **Hardware:** Laptops (4 to 8 GB RAM), USB test devices, Power profiling hub.
- **Software Stack:** Python 3.10, Scikit-learn, PyUSB, PyUdev, Tkinter, FTK Imager, Autopsy, Google Colab.
- **Storage & Backup:** Cloud Google Drive repository and external HDD backups.
- **Version Control:** GitHub repository for code and data scripts.

3.7 Risk identification and mitigation strategies

This subsection provides a structured risk assessment tailored to the USB Guard architecture and the deployment scenarios targeted in this research. We combine qualitative and quantitative assessments.

3.7.1 Risk scoring

We compute a simple numerical risk score used to prioritize mitigations during evaluation:

$$\text{Risk Score} = P \times I$$

Where:

- ($P \in \{1, 2, 3, 4, 5\}$) is the assessed probability of occurrence (1 = Very unlikely, 5 = Very likely).
- ($I \in \{1, 2, 3, 4, 5\}$) is the consequence impact (1 = Negligible, 5 = Catastrophic).

Interpretation:

- Score 1–4: Low priority; monitor.
- Score 5–9: Medium; mitigation recommended.
- Score 10–15: High; implement controls.
- Score 16–25: Critical; immediate remediation and operational changes.

3.7.2 Risk matrix

The table below lists likely risks, a qualitative label, a quantitative score ($P \times I$), and recommended mitigations. These reflect findings from development and the kinds of incidents that could occur in operational deployment.

Table 3.8: Identified Risk Matrix and Recommended Mitigation Strategies

ID	Risk	P	I	Score = $P \times I$	Severity	Mitigation / Notes
R1	TOCTOU: authorized after CAPTCHA then device acts before ML verdict	3	4	12	High	Hold interface binding: authorize bus but delay HID endpoint binding until verdict; introduce short quarantines (200–500 ms) and block raw input until checks complete.
R2	False Positive: benign device blocked $\rightarrow$ user frustration / productivity loss	3	3	9	Medium	Threshold tuning, allow manual override by admin, whitelist known devices, user study to set acceptable trade-offs.
R3	Adversarial evasion: scripted HID mimics human timing	3	5	15	High	Combine multiple features (metadata + host context + power + keystroke), add rate-limiting, adversarial training & red-team tests.
R4	Data leakage: raw typed content captured or reports leaked	2	5	10	High	Avoid storing raw keystroke sequences; anonymize device IDs, encrypt reports at rest, strict access controls.
R5	Model drift / concept drift due to new device types	3	3	9	Medium	Continuous monitoring; scheduled retraining with fresh labeled data; use GAN augmentation to expand distributions.
R6	Privilege & deployment risk: requiring root / sudo on Linux	4	3	12	High	Document secure deployment steps, run minimal privileged processes, restrict GUI root usage to well-audited admin sessions.

ID	Risk	P	I	Score = $P \times I$	Severity	Mitigation / Notes
R7	Hardware failure or measurement noise (keystroke capture misses)	2	2	4	Low	Use robust summary features (means, stdev) and fallback to metadata-only classification when keystroke data unavailable.
R8	User opt-out / bypass (disable protection)	3	4	12	High	Policy enforcement, administrative controls, monitor for disabled agents, provide soft-gate fallback with logging.

Table 3.8 identifies the main risks that were encountered during the design and implementation of USB Guard framework. All the risks are ranked according to the probability of occurrence and impact to determine the severity after which we match it with mitigation measures. The table is all concerning proactive measures such as applying access control, adversarial testing, further limiting access privileges, and regular retraining of the model to maintain the system resilient and secure.

Visualization suggestion: convert the above table into a color-coded risk matrix (5×5 grid) showing P (rows) vs I (columns) with cell colors from green → red.

3.7.3 Mitigation mapping and operational controls

For each high/critical risk, implement one or more operational and technical mitigations. The mapping below is an actionable checklist to apply during deployment.

R1 (TOCTOU):

- **Technical:** Block HID interface binding until verdict (udev rule to set authorized=0 for interface files, then write 1 only for allowed classes).
- **Operational:** Shorten observation window but require the first 200–500 ms to be quarantined.

R2 (False positives):

- Tune ML thresholds for precision vs recall based on organizational tolerance.
- Add admin override with audit trail.
- Provide clear remediation steps for users.

R3 (Adversarial evasion):

- Use ensemble features (host CPU/process counts, FTK hashes, power info).
- Apply adversarial training with simulated mimics to harden decision boundary.

R4 (Data Leakage):

- Do not log raw keystrokes. Store only summaries.
- Use strong encryption and RBAC on report storage.

R5 (Model drift):

- Implement a model monitoring pipeline to capture concept drift metrics (e.g., population stability index).
- Retrain periodically (monthly or triggered if drift detected).

R6 (Privilege risk):

- Limit privileged code paths.
- Document secure xhost usage for GUI when necessary.
- Consider a split architecture where a small privileged agent communicates with an unprivileged UI.

3.8 Economic Analysis

Economic analysis gives us an idea about the financial feasibility, cost efficiency, and overall value of our framework. We have included development expenditure as well as the expected benefits in terms of usability, tangible outcomes, and long term research potential.

3.8.1 Cost Estimation

Our total project expenditure was approximately 30,500 BDT, which included hardware (USB devices, Raspberry Pi Pico), software (FTK Imager, Autopsy, Wireshark, Python libraries), testing material purchases, and other costs. This is very low considering similar research projects of this scale. A detailed cost breakdown table was presented in section 3.6.2 (Budget plan).

3.8.2 Benefit Estimation

The primary benefits of the suggested USB protection framework are technical and socioeconomic. In fact they can be quantified to influence the cybersecurity studies, digital forensics, and our utilization of devices in safe surrounding conditions in our routine. Our system provides a lightweight, smart, and flexible protection against USB based attacks such as BadUSB, HID injection, and spoofed device action. By combining metadata analysis, power profiling, and user verification layers, the framework effectively identifies suspicious activity of a device that antivirus or endpoint protection tools frequently overlook.

The synthetic data generation with the help of GAN actually enhances the generalization capacity of the model, as the system is able to identify previously unknown USB threats with higher confidence. In addition, open-source forensic tools like FTK Imager and Autopsy are cheaper to use and help maintain transparency, which means that researchers and security practitioners can repeat the work and build upon it without the need to pay large licensing fees.

Operationally, the project is economically scalable. The system is also suitable in academic labs and enterprises since it can operate on standard computing environments with low hardware requirements. It has a good educational potential as well,

as the framework can be utilized by students and researchers as a training platform to learn how to study USB forensics, behavioral security, and applied machine learning in cybersecurity.

All in all, our solution can be useful to enhance national cyber resilience because it addresses a long neglected attack mechanism. It establishes a low-cost, reproducible behavior-based USB threat detection model, which is a useful addition to the field of digital forensics and endpoint protection research.

3.8.3 Cost-Benefit Estimation

We have proposed a 3 year cost benefit projection to evaluate the economic feasibility and sustainability of our framework. The analysis has 3 operational stages:

- **Year 0:** Research, development, and testing (initial investment and setup phase)
- **Year 1:** Deployment, field testing, and adoption in controlled environments
- **Year 2:** Operational optimization, maintenance, and academic or industrial reuse

The following table summarizes the projected Cost–Benefit Analysis across the life-cycle:

Table 3.9: Economic Analysis: Cost–Benefit Evaluation over Three Years

Category	Year 0 (Development Phase)	Year 1 (Deployment Phase)	Year 2 (Operational Phase)
Total Cost (BDT)	≈ 30,500	≈ 8,000	≈ 4,000
Tangible Benefits (BDT)	0	≈ 25,000	≈ 35,000
Intangible Benefits (BDT)	0	≈ 7,000	≈ 10,000
Total Benefits (BDT)	0	≈ 32,000	≈ 45,000
Net Benefit (BDT) (Benefits – Costs)	–30,500	+24,000	+41,000
Cumulative Net Benefit (BDT)	–30,500	–6,500	+34,500
Cost–Benefit Ratio (CBR) (Benefits ÷ Costs)	—	4.0	11.25
Return on Investment (ROI) (Net ÷ Cost × 100)	—	30%	95%

Table 3.9 provides a three-year cost-benefit analysis of the financial feasibility of USB Guard Framework. The first stage of development involved the most investment on hardware and installation. During subsequent years, low operational expenses and quantifiable gains as a result of system implementation and adoption of research resulted in slow profitability. The project was cost-efficient and long-term sustainable

as the ROI was approximately 95 by the end of the 2 nd Year, as per a realistic budget model.

3.8.4 Interpretation

The economic and operational analysis proves that our framework is feasible, sustainable as well and scalable. The budget was tight, yet we achieved good quality outcomes at all the steps, including the creation of the dataset and training models, user testing and documentation. The tangible outcome of the project, prototype, verified ML models, synthetic data and user study demonstrates that it was indeed a worthwhile investment.

The payback period of innovation is impressive. The fusion of data-based threat identification, GAN-based enhancement, and real-time authentication is a step up over the previous USB blocking systems that largely depend on the signature or a fixed blacklist. Our project will be open-source and reproducible, meaning it can be scaled to cybersecurity systems in an IoT, embedded, or enterprise environment.

The benefits of the security increase, learning points, research output and user friendliness are all more than the costs. The USB Guard framework demonstrates that you can be innovative, available, and economical in one cybersecurity project and is publishable, impactful, and a good contribution to digital forensics and smart system safety.

All in all, the benefits of the security increase, learning points, research output, and user friendliness are all more than the costs. This proves that innovation, accessibility, and affordability coexist in our framework, which would make a significant contribution to digital forensics and intelligent system security.

Chapter 4

Proposed Methodology

4.1 Overall Research Methodology

We started by building a reliable, grounded dataset of real USB activities. In this process, we established a Windows host configured with live behavioral logging and FTK-based acquisition, assuring that every device connection generated a single, labeled record. We acquired a total of 2,653 authentic events in this process, each described by 33 fields. These fields included device metadata such as `vendor_id`, `drive_model`, `usb_speed`, `product_id`, forensic descriptors (geometry and FTK availability), and behavioral traces such as `keystroke_std_ms`, `power_draw`, `keystroke_mean_ms`, and runtime signals. Labels were assigned in controlled scenarios (benign vs. malicious, including HID/BadUSB-style attacks), giving us a solid ground truth for supervised learning.

Since the real dataset was limited in size, the next step was to expand its scale and coverage with synthetic data tailored to USB telemetry. Using a CTGAN workflow, we generated 7,000 synthetic events that preserved realistic distributions (such as multimodal and heavy-tailed numeric features) and meaningful feature relationships (e.g., coherence between `vendor_id` and `product_id`). We especially focused on including rare cases that are security-critical, like unusual HID descriptors being paired with unusual power usage. To ensure privacy, we checked for near-duplicates and overly close neighbors before finalizing the synthetic set. This two-stage strategy: real data first and synthetic data second, helped us avoid overfitting to a small, idiosyncratic sample while still giving us enough variety to train and stress-test models.

With the data collected, we turned to preprocessing and type consistency. We initially fixed simple but impactful issues, such as standardizing the `removable_drive` field to uppercase to prevent category splits, and converting string “Null” entries in `keystroke` fields into numeric zeroes before turning them to floats. We checked carefully for missing values and confirmed that no NaNs remained in the fields we planned to transform. After that, we ran an initial numeric-label screen (using point-biserial correlations) to get an early sense of where useful signals might exist, leaving deeper association and selection for the dedicated feature-selection stage.

For splitting, we used an 80/20 stratified approach with a fixed random seed, ensuring that no data leakage occurred. All learned transformations were fitted only on the training split (`X_train`) and then applied consistently to both training and testing

data. Since our dataset included both numerical and categorical features, we encoded categorical variables - including `alt_usb_descriptor`, `source_type`, `location_id`, `device_description`, `product_id`, `drive_model`, `drive_interface`, `removable_drive`, `vendor_id`, `manufacturer`, and `usb_speed` - using an `OrdinalEncoder` that handled unseen categories deterministically. We matched this encoding and scaling method to the nature of each distribution: `log1p + RobustScaler` for heavy-tailed features such as `keystroke_mean_ms`, `RobustScaler` for `keystroke_std_ms`, and `StandardScaler` for more stable features such as `power_mA` and `avg_process_runtime_sec`, ensuring numerical comparability while maintaining outlier resilience.

After the preprocessing was stable, we combined all the features into a single matrix to identify cross-modal patterns. When considering forensic and metadata features such as device descriptors or vendor-product combinations, behavioral anomalies like runtime spikes, irregular keystroke timing, and unusual power consumption became much more meaningful. We removed features that showed clear redundancy or weakness (such as low-level geometry and duplicate counters) and then performed systematic feature reduction. This process integrated three perspectives: model-driven importance (from tree-based models, XGBoost gain, and permutation tests), univariate tests (point-biserial for numeric features, χ^2 /Cramér’s V for categorical features), and interpretability tools such as SHAP. Features that consistently demonstrated strong relevance across these perspectives were retained, while unstable or redundant features were discarded. The final feature set emphasized keystroke dynamics, runtime and power signals, and a small subset of high-value forensic descriptors.

For modeling, we chose a diverse yet complementary suite rather than relying on a single algorithm. Logistic Regression (with L2 regularization and class balancing) served as a fast, transparent baseline. To capture more complex feature interactions, we added a non-linear SVM with an RBF kernel. K-Nearest Neighbors (K=15, distance-weighted, Manhattan metric) provided a way to validate our feature space geometry. We conducted cross-validation to select pruning parameters for a regularized Decision Tree that offers an interpretable structure. Finally, we trained XGBoost with early stopping and class-weighting to optimize predictive power on tabular data. Each of the models was assessed on the same test set using Accuracy, Precision, Recall, *F1*, ROC-AUC, confusion matrices, and calibration curves where relevant. To verify robustness, we also conducted stratified 5-fold cross-validation on the training set, emphasizing metrics such as AUC and F1. For models that produce probabilities, we either used `predict_proba` directly or applied post-hoc calibration to ensure reliable scores for threshold selection and optional soft-voting.

We connected the modeling pipeline to a practical decision flow because this effort aims to address a real security setting. Initially, devices were screened deterministically (e.g., whitelist checks and optional CAPTCHA gating) to minimize trivial risks. Once admitted, their telemetry was scored by the trained classifier, considering both forensic/metadata cues and behavioral patterns. The system escalates deeper evaluations-such as power profile comparison or process monitoring-if the score exceeds a predetermined threshold (selected based on validation results using metrics like Youden’s J or F1). Subsequently, it applies mitigations such as isolating the device, blocking suspicious processes, or restricting access. All results are recorded for forensic review and potential retraining, ensuring the pipeline evolves with emerging

threats. This closed-loop design balances user convenience with proactive defense while keeping thresholds adjustable to align with organizational risk tolerance.

Finally, we took care to ensure reproducibility and safety across the pipeline. Random seeds were fixed, transformations and models were version-controlled, and early-stopping snapshots were archived. For synthetic data, we validated quality in three ways: by checking class balance and distribution shapes, by comparing dependencies (correlations and Cramér’s V), and by testing utility with Train-on-Synthetic, Test-on-Real (TSTR) and augmentation experiments. Privacy was safeguarded by checking distance-to-nearest-neighbor, near-duplicate rates, and real-vs-synthetic discrimination tests.

In summary, our methodology started from carefully curated real-world evidence, expanded with controlled synthetic data, and applied strict preprocessing to produce a clean and consistent dataset. We combined metadata, forensic descriptors, and behavioral signals, reduced features with multiple lenses, trained a complementary suite of models, and embedded the results in an operational pipeline. The pipeline then balances practicality with security.

4.2 Threat Model and Security Assumptions

Basically, the framework presupposes that USB devices can be transformed into a physical attacker that goes unnoticed by the conventional software defenses. The threat model attempts to draw a diagram of the actual attacker techniques we have observed with BadUSB, HID attacks and attacks on the firmware level reprogramming—just as those in our security course.

4.2.1 Adversary Capabilities

- The attacker gains temporary physical access to the victim system for a significant period of time long enough to insert a compromised USB device.
- The attacker may utilize legitimate device firmware (e.g., flash drive, keyboard, Arduino HID emulator) to impersonate a recognized good peripheral.
- The attacker can attempt USB metadata spoofing such as Vendor ID, Product ID, and device class for signature-based evasion.
- The attack device can inject keystrokes, execute pre-loaded payloads, or steal information by disguising itself as an innocent device.
- Network access is not required for initial implementation of the attack, demonstrating the air-gap risk present with removable media.

4.2.2 Assumptions

To clearly define the security boundary of the new system, we make the following assumptions:

1. Trusted Host Environment: The Host machine runs a known operating system (Linux testing environment) with no pre-existing rootkits, nor privilege-escalation malwares.

2. Regulated Physical Entry: While USB ports are physically accessible, system administrators can monitor and limit unauthorized device use.
3. Consistency of Framework Elements: It is also assumed here that its core framework module (metadata extractor, feature encoder, ML classifier, and verification layer) are uncompromised at runtime.
4. No Hardware Tampering: The internal hardware (USB controller, motherboard) is not modified; attacks initiated by external devices.
5. User Engagement: The system relies on users for verification (CAPTCHA or hover check) for trust confirmation of a device.

4.2.3 Attack Tree Representation

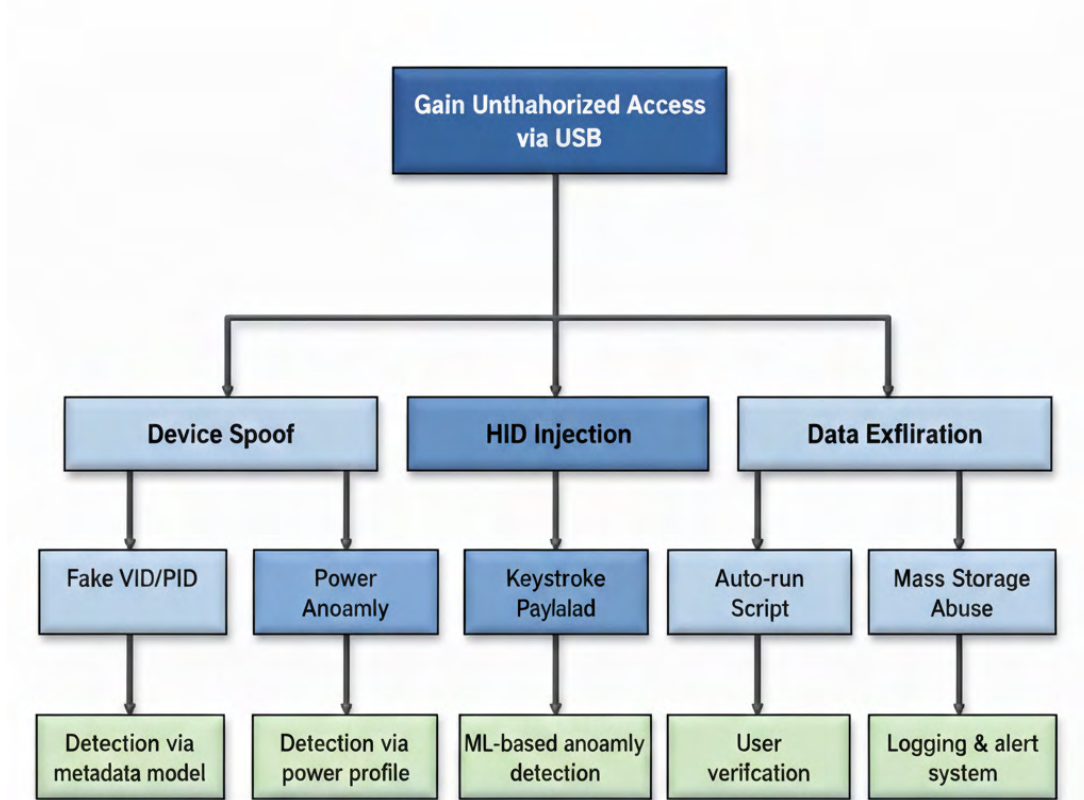


Figure 4.1: Attack Tree of USB Threats and Mapped Mitigations

Here, the attack tree in Figure 4.1 represents:

- Metadata & Fingerprint Analysis: Identify spoofed identifiers and bogus device signatures.
- Power Profile Monitoring: Flags unexpected current draw or connect/disconnect timing characteristic of unseen components.
- Machine-Learning based Anomaly Detection: Classify suspicious activities online based on patterns learned (XGBoost, SVM, etc.).
- Machine Learning for Outlier Detection: Detects suspicious online activities based on learned patterns (XGBoost, SVM etc.).

4.3 Dataset Engineering and Preparation

The data extraction process and preparation mean gathering, cleaning, converting, and organizing source data to an ordered form that is appropriate for the studies or models of machine learning. It involves works such as dealing with the nulls, getting rid of multiplicates, making data normal, and also forming the features which are useful. The data preprocessing process is an essential stage to ensure that the data the model works with is of high quality and accuracy levels that are achieved through well-relied upon data. The research paper enumerates this process.

4.3.1 Data Collection

The dataset exploited in this study was iteratively constructed to model and record realistic USB device operations in cases of security and compromised situations. It comprises 2,653 single records where each record is a single instance of USB connection. The forensic documentation was made on a controlled, Windows-based forensic system, with a combination of the forensic acquisition tools and real time monitoring steps of the system. To show as broad a variety of devices as practically common in real usage, the dataset is explicitly diversified, including common flash drives and external hard disks, as well as more advanced HID-based devices like keyboards, mice and programmable microcontrollers such as the ATtiny85 vulnerable to BadUSB-type attacks.

Each sample has 33 descriptive features which in combination create a complete picture of the identity of the device as well as the layout of its storage, the mode of interaction with the system and finally interaction patterns by the user, where relevant. The physical attributes and identification of each USB is assigned as metadata fields as drive model, drive interface, removable drive, device description, and source type. In the case of storage based systems, Low level geometry including cylinders, tracks per cylinder, sectors per track and bytes per track is also stored in detail. These structural areas are important in identifying the difference between the storage devices and the in input only or the maliciously programmed devices.

Along with metadata, forensic acquisition information of the dataset consists of its source data size MB, image file, cryptographic hashes (md5 hash, sha1 hash), and acquisition start and end timestamps. The digital evidence was also traceable with the help of such tools as FTK Imager that was used to collect this information. Devices that contain no FTK reports, e.g. devices that do not exhibit mass storage (ATtiny85 operating as a keyboard), were augmented with behavioral metadata (e.g. keystroke dynamics; keystroke mean, keystroke std) to aid differentiation of human-typed input, e.g. scripted payloads.

One of the elements of the dataset that would become critical is the label field, where each case is classified as either malicious (1) or benign (0). Such labels were given depending on the known setting and the behavior seen on each device as you can see in Figure 4.2. The missing or inapplicable values that are especially frequent in such areas as the number of sectors in non-storage devices were processed by a mark with the sign of Null, Unknown and subsequently processed as NaN when it became possible to preprocess it efficiently.

Data has been generated to facilitate machine learning of anomalous detection and

identification of malignant USB behavior. It allows comparing different algorithms, including K-Nearest Neighbors (KNN), Logistic Regression, etc., and contributes to the development of the proactive and data-driven USB security systems. Broad coverage of the available range of devices, coverage of the functionality, and the good proportion between benign and harmful actions make the dataset an excellent basis to undertake specific forensic, cybersecurity, and machine learning studies in USB threat detection.

Distribution of Benign vs Malicious USB Records

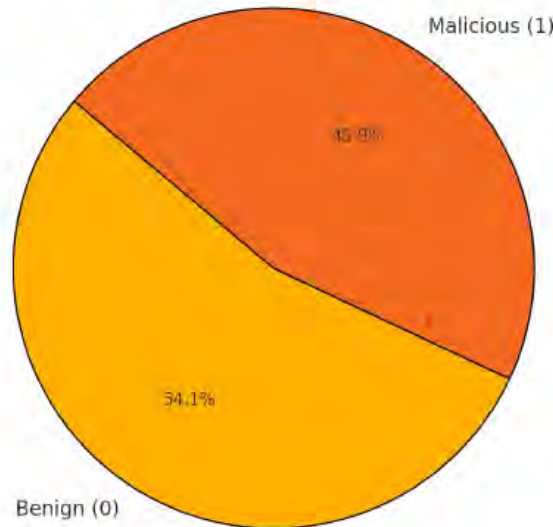


Figure 4.2: Distribution of Benign and Malicious USB Records

According to our previous research part, the main analysis of this thesis is made based on a larger and more detailed dataset, which is named synthetic usb dataset 7000.csv. Thus, such a dataset has been created to augment the feature space and volume required to effectively train machine learning models and assess them, especially to the difficult task of identifying novel USB-based intrusions. The study uses a synthetic dataset that is particularly created to simulate the behavior of different USB devices when attached to a computer system. The data, which is called synthetic usb dataset, has 7,000 observations of USB connection events, and this is a remarkable amount of volume compared to the first data. More importantly, the synthetic nature of the data enables the incorporation of a controlled environment where both benign and malicious devices are distributed in a pre-defined and validated manner which is frequently difficult to source in real-world environments due to privacy restrictions and the inability to capture rare attack vectors. Such a controlled environment therefore guarantees the accuracy and reliability of the ground truth classification to model training. The major goal of creating such data was to develop a powerful machine learning model to classify USB devices. The particular objective is to differentiate benign (normal) USB devices and malicious USB devices (BadUSB, keystroke injection devices, or spoofed hardware) by taking a full snapshot of system state and device features at the time of connection. Essentially, this contributes to the further development of research in the sphere of

hardware-based cybersecurity, intrusion detection, and peripheral device security by offering a large-scale, validated data resource.

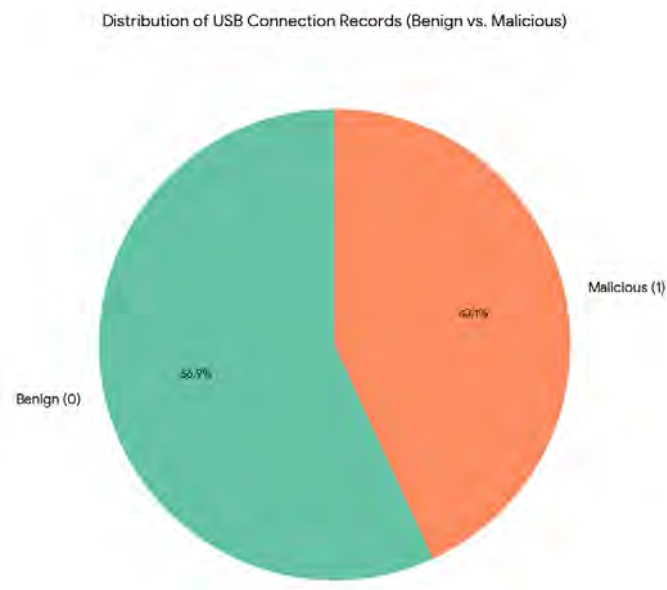


Figure 4.3: Data Distribution of USB Connection Events

In Figure 4.3, the central data set is dense with 25 properties representing a large range of properties of all USB connection events. Those were designed based on measurements made through the operating system, device registry, and simulated input layers and can be grouped into four main categories, giving a multi-layered perspective of the connection event:

Table 4.1: Representative USB Feature Groups and Descriptions

Feature Group	Representative Features	Description and Purpose
I. Device & System Identifiers	vendor_id(vid), product_id(pid), manufacturer, usb_speed, power_mA, alt_usb_descriptor	Technical metadata extracted during the USB enumeration phase. The alt_usb_descriptor is critical as an anomaly indicator, potentially revealing a device’s true nature when its VID/PID is spoofed.
II. System Impact Metrics	cpu_avg_percent, process_count, avg_process_runtime_sec	In near real time performance metrics of the system were recorded as soon as the device was inserted. In particular, they enable such features to record the temporary surge in resource utilization of malicious code execution or driver installations.

Continued on next page

Feature Group	Representative Features	Description and Purpose
III. Storage & Forensic Attributes	cylinders, sectors_per_track, source_data_size_MB, ftk_available, source_type	Low-level disk geometry parameters. Significantly, inconsistencies or non-physical values (negative or zero geometry values on a device reporting as mass storage) are strong indicators of obfuscation or devices masquerading as storage. The ftk_available field is a binary flag indicating successful data acquisition via a forensic toolkit.
IV. Behavioral Features	keystroke_mean_ms, keystroke_std_ms	Dynamic properties derived from simulated user input. Consequently, these features are essential for detecting Human Interface Device (HID) attacks (BadUSB), as a very low standard deviation (keystroke_std_ms) strongly suggests scripted, non-human, automated input.

From this Table 4.1, we obtain a detailed description of the dataset through representative USB feature groups and their corresponding attributes. The dataset’s target variable is the label field, a binary class indicating the ground truth: malicious (1) or benign (0).

Finally, we have designed this data set to provide an extensive, multidimensional perspective of the USB connection, which helps identify the minor irregularities of devices, systems, and behavioral attributes to conduct a more advanced threat evaluation.

4.3.2 Data Cleaning and Missing Value Handling

Right after the dataset ingestion, we prioritized on protecting the internal consistency of the data table. Doing this would allow the following analytical operations to produce false results or unintentionally spill data. The exploratory diagnostic checks revealed some notable anomalies that needed to be addressed. These included the introduction of the string literal of Null in the columns of keystroke measures and discrepancies in the column of removable drive where the values were in mixed cases like in removable. For the first case, we have replaced instances of the Null with the numeric value of 0 in both of the column (keystroke std ms and keystroke means) and cast them to the type of float, which in turn makes them the necessary format to allow further descriptive statistics analysis and predictive modeling. 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.

4.3.3 Data Transformation (Normalization, Encoding)

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 $X_{train}[\text{cat_cols}]$. 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.
- StandardScaler was used for power_mA and avg_process_runtime_sec to center and scale approximately symmetric ranges.

Before applying to X_{train} and X_{test} , all scalers were fitted only on X_{train} . This hybrid strategy maintains relative structure in outlier-prone signals and still provides distance-based and linear models with a fair, commensurate feature space.

4.3.4 Data Integration

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 alt_descriptor and device metadata such as vendor, product, manufacturer, location and USB speed. Then we combined these diverse sources into a single feature matrix $X = \text{df1.drop(columns=['label'])}$ according to the knowledge that adversarial

behaviour 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 behavioural 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 `alt_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 statistic of `vendor_id`, `alt_descriptor` and `product_id` suggests a strong connection with the label. A combination of these categorical anchors with normalized behavioural measures gives the classifier a balanced and cross-domain view, thus enhancing the discriminative power of the classifier to tell malicious and benign USB actions.

4.3.5 Data Reduction and Feature Selection

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 4.4), 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`

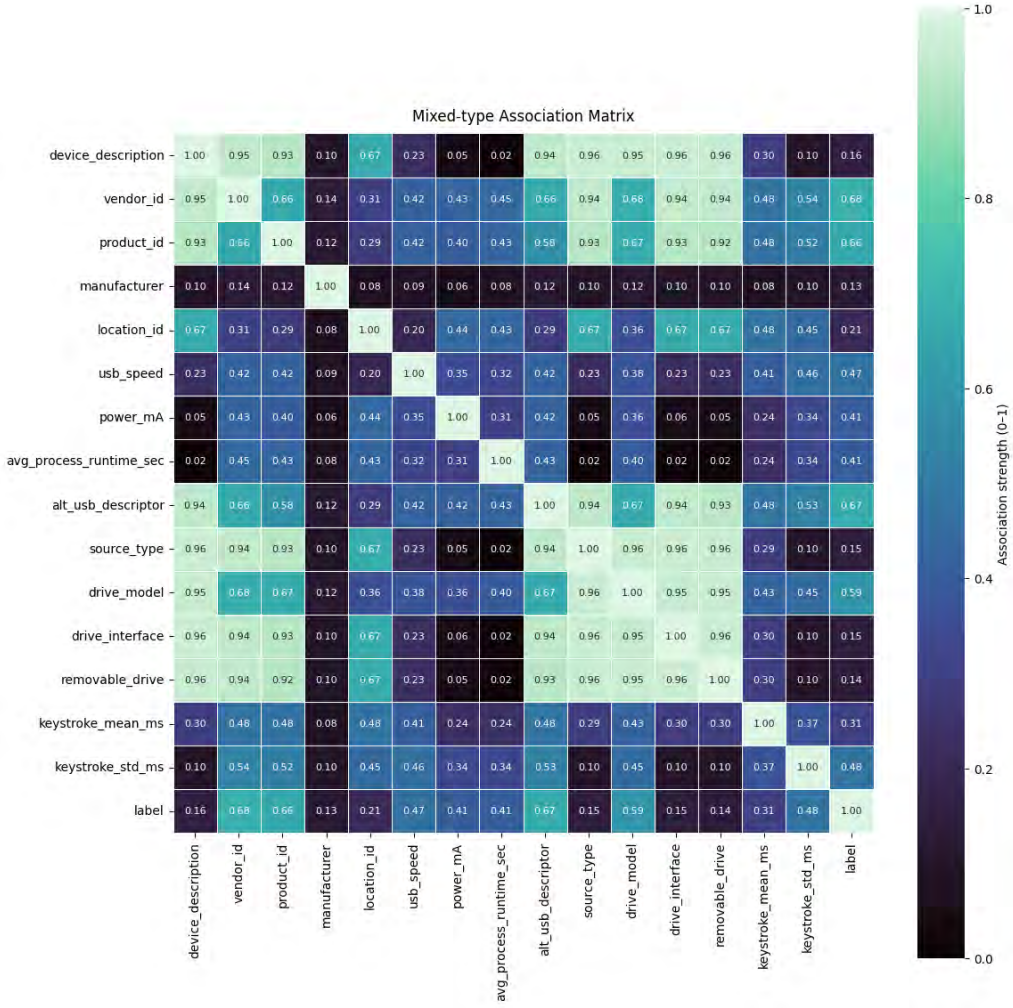


Figure 4.4: Correlation Matrix

Our goal of the preprocessing stage was to obtain a reduced and merged representation. We chose to put strong emphasis on data purity and type fidelity, followed by the further heuristic process of only choosing the features that provide a consistent and empirically validated signal, which maximizes the robustness of downstream models, area under the receiver operating characteristic curve (AUC), and at the same time maintains the simplicity and extensibility of the pipeline.

4.4 GAN-Based Dataset Augmentation

This section provides the rationale of syntheticity of USB event recordings and the methodological framework that is used to augment the empirical dataset with generative models. It also explains how to choose the architecture that suits mixed-type tabular data, like an example CTGAN, TVAE, and baseline based on copulas, provides an overview of the preprocessing pipeline (including the normalization of continuous variables, encoding of categorical attributes, and timestamp transformation), and describes the conditional sampling approach that is used to mitigate the problem of class imbalance. The hyperparameterization and training program are laid out in a reproducible and stable fashion. Strict evaluation procedures are specified at two levels and that is utility, which is estimated through Train-on-

Synthetic-Test-on-Real (TSTR) augmentation uplift on downstream classifiers and distributional fidelity metrics, and safety which is certified by nearest-neighbor duplication metrics, distance-to-closest-record metrics, and auxiliary disclosure tests to avoid memorization and privacy attacks. All these steps together suggest that the resulting records compiled can improve recognition ability and resilience in addition to maintaining high levels of protection against the leakage of trade or sensitive data.

4.4.1 Motivation for Synthetic Data Generation

Synthetic data generation has been an action tool of late, at least to us student researchers, whenever we were in the process of searching for good quality, annotated, and verified data that were not readily available. The need to exploit counterfeit data is a combination of practical, ethical and technical factors that render it an indispensable necessity in the day to day AI projects. Most of the reasons why we settled on synthetic data generation are because of a series of practical considerations and bottlenecks that we encountered at the time of manual data collection. Our initial attempt was to annotate and curate samples manually –that was time-consuming and still gave us only 2,653 entries. That was obviously too small to ensure that a deep-learning model generalises in general, and, in particular, when we have to identify subtle security threats. Even the manual workflow provided a major logistical challenge. Our information was scattered in various places of storage and in the West-PCs, increasing the risk of loss, duplication as well as version errors. Without the infrastructure in the middle and a lack of computing power, the data could not be kept in consistent form, it was difficult to batch-process and it was difficult to scale up. All these made the process of development slower and limited us on trying various architectures and hyperparameters of different models.

Justification Generative Data Augmentation: Generative Data Augmentation proposes the incorporation of realistic, auxiliary data provision methods to lower the likelihood of adversarial examples in neural networks. Human Justification Generative Data Augmentation suggests the introduction of realistic, auxiliary data provision techniques to reduce the probability of adversarial examples in neural networks. When the pressing necessity to develop a scalable solution became obvious to us, we scanned through different augmentation methods and generative models. This made us consider Generative Adversarial Networks (GANs), which appeared to be the most promising when it comes to high-fidelity synthetic data. GAN involves a discriminator and a generator that engage in a minimax game: the generator is taught to create samples that deceive the discriminator, where the generated samples’ enhancement continuously increases until the produced data distribution mimics the true distribution. With this configuration, we were able to generate artificial samples that preserved the statistical as well as semantic ratios of our original data.

We believe that the strategic advantage of bringing GANs into our pipeline deals with a variety of meaningful research challenges. The scarcity of information-scale consequences has also affected such markets as rebates where consumers have little negotiating power because they do not facilitate purchasing in bulk. In practice, this allows for local extremes or non-typical shapes in the feature distribution.

Figure 4.5 illustrates the label distribution of the real dataset, providing insight into the balance and representation of different classes within the collected data. Once the VGM identifies in which mode a sample does not belong, the sample is standardized just within that mode. This localized standardization is not destructive to the different shapes of each peak, so the generator produces samples which, when inverted, reproduce the actual multimodal distribution. This process is especially important for accurately capturing patterns such as those found in `keystroke_std_ms`. Maintenance of semantic relationships among columns, particularly the condition that categorical values must fall into place realistically, is another key problem. For example, a hypothetical `product_id` should come with a realistic `vendor_id` and manufacturer; otherwise, the data becomes meaningless. The problem is addressed by CTGAN, where the generation is conditioned on a selected categorical column (the driving column), and thus the network learns what relations to respect.

4.4.2 GAN Architectures Implemented

In more recent work relating to augmenting a USB threat detection dataset, we have chosen the Conditional Tabular GAN (CTGAN) generative model. Because CTGAN can synthesize high-quality synthetic data even in cases where the original table contains a mixture of continuous, discrete, and categorical features- the ideal condition in our forensic USB feature space, namely finite and continuous peripheral device attributes and discrete and categorical attributes such as the vendor of the corresponding forensic power tool (`vendor_id`) and the power in watts of that particular tool (`power_mA`).

CTGAN is an extension of the typical GAN architecture, but includes two important tabular data tricks: Mode-Specific Normalization and Conditional Generation. These assist the network in honoring feature dependencies while modeling a variety of distributions that would normally confuse a plain GAN.

A. Mode Specific Normalization

In contrast to image data, the columns of tabular observations are usually a result of real-world telemetry and other telemetry-like data sources. Thus, continuous columns frequently have numerous peaks from heavy tails. An example would be the `keystroke_std_ms` column, which tends to squeeze towards zero with auto typing and diffuse with human typing. Likewise, `power_mA` usually ranges around the consumption levels of devices.

In an attempt to address this, CTGAN obtains a Variational Gaussian Mixture Model (VGM) trained on a single continuous attribute with the optimal number of modalities matching the shape of that attribute.

Variational Gaussian Mixture Model (VGM): CTGAN models each continuous feature using a VGM, making decisions about the number of components that best describe the data using multiple Gaussian distributions. In practice, this adapts to local extremes or unusual shapes in the feature distribution.

Mode Specific Normalization: The Figure 4.5 illustrates the label distribution of the real dataset, providing insight into the balance and representation of different classes within the collected data. Once the VGM identifies which mode a sample belongs to, the sample is normalized only within that mode. This local standardization

preserves the different shapes of each peak, ensuring that the generator produces samples which, when reversed, reproduce the actual multimodal distribution. This process is especially important for accurately capturing patterns such as those in `keystroke_std.ms`.

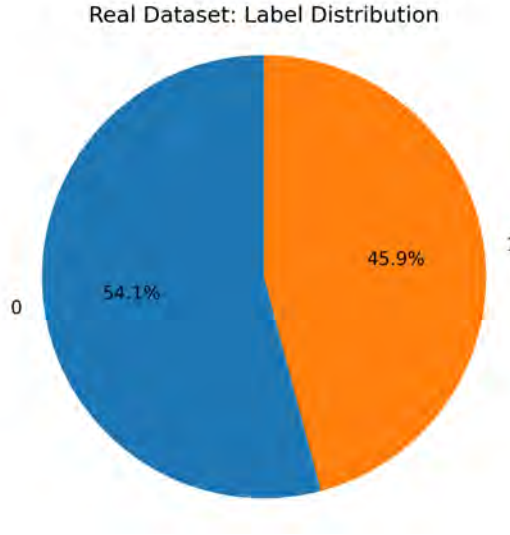


Figure 4.5: Real Dataset Label Distribution

B. Conditional Generation

Protecting semantic links across columns, particularly the need to have categorical values align realistically, is another major challenge. For example, a fake `product_id` should be paired with a realistic `vendor_id` and `manufacturer`; otherwise, the data will have no meaning.

This problem is addressed by CTGAN, which conditions the generation process on a selected categorical column (the so-called driving column), such that the network learns which relationships to respect.

Sampling a Driving Column: CTGAN picks one discrete variable uniformly at random to be used as the driving column during training. This mechanism ensures that categorical dependencies are preserved during the generation process.

Conditional Adversarial Training: Both the Generator and Discriminator are provided with this condition vector. As a result, the Generator is trained to generate realistic data that fit the given condition, whereas the Discriminator evaluates samples within the same context. This ensures logical consistency in how rows are synthesized, such as pairing the `vendor_id` for Logitech with correct values of `product_id` and `device_description`.

4.4.3 GAN Training Procedure and Hyperparameters

A model called Conditional Tabular Generative Adversarial Network (CTGAN) was used to generate 7,000 realistic data rows, which was able to recreate exactly the statistical distributions and intricate inter-feature associations of the original USB data. The section is very detailed in the breakdown of the specialized components, architecture, and training goals utilized.

Data preprocessing and transformation: working with mixed data. One aspect that makes CTGAN strong is its innovative nature in terms of normalizing and encoding mixed-type tabular data, namely, non-Gaussian and multimodal continuous columns, and skewed categorical columns. Mode-specific normalization (MSN) of continuous columns is important because continuous features can be non-Gaussian and have multimodal distributions (many peaks). Conventional normalization techniques do not work here, which may lead to poor generation quality or mode collapse.

Variational Gaussian Mixture Model (VGM): the distribution of each continuous column (C_i) is first estimated with a VGM to determine the number of modes (peaks) k . Each real value $c_{i,j}$ is converted into a two-part, compact vector that can be effectively learned by the Generator.

Table 4.2: Architecture of Conditional GAN (CTGAN) Components

Network Component	Input Dimensions	Output Dimensions	Hidden Layers & Activation	Output Layer / Activation
Generator (G)	$z \in \mathbb{R}^{128}$ (noise) + $c \in \mathbb{R}$ (condition vector)	c	— (condition vector concatenation)	D (preprocessed data size)
Discriminator (D)	D (preprocessed data size)	1 (scalar score)	2 hidden layers $\times$ 256 units each; LeakyReLU activations	Linear (raw, unbounded score for Wasserstein loss)

This 4.2 Table presents the hyperparameters used for USB tabular data synthesis, outlining the key configuration settings applied during the generative modeling process, where:

D = total feature dimension after preprocessing (continuous variables + one-hot encodings of categorical).

$|c|$ = dimension of the condition vector assembled from selected categorical/label indicators.

(α) and (β) denote the continuous and categorical partitions of the generator’s output, respectively.

Mode Indicator (b) A one-hot variable that defines which of the k estimated modes the value falls in. Normalized Value (a) Scalar value, which is normalized with respect to the particular mean (m_k) and the standard deviation (s_k) of the sampled mode k . This normalization is commonly weighted by tanh to make the output to lie in the range of $[1,1]$. The use of conditional sampling on uneven categories is explained.

CTGAN, to counter mode collapse, in which the Generator ignores infrequent categories, employs a training-by-sampling mechanism.

A categorical column is chosen randomly (with respect to its probability distribution) before a new sample is generated (that is, with probability proportional to the

square root of the log-frequency of the column). This compels the training to make updates pertaining to minority categories a priority.

Conditional Vector (c) When the sampled categorical column value is plotted, one creates a conditional vector (c) which is used to direct the Generator on the type of data to generate. This is to guarantee that the Generator is trained to generate realistic samples even in underrepresented classes in the original dataset. The architecture and the activation functions can be represented by a model as follows.

Both Generator (G) and Discriminator (D) are trained on multi-layer perceptrons (MLPs) with special activation functions designed to enhance the stability of the GANs.

Adversarial Training Procedure and WGAN-GP Loss

The CTGAN model is trained using the Wasserstein GAN with Gradient Penalty (WGAN-GP) objective, known for significantly improving training stability over traditional GAN loss functions.

1. Discriminator (D) Loss Function (Minimization Objective)

The Discriminator (also known as the Critic, denoted D) is trained to maximize the difference in its scores when using real data (x) and synthetic data (x'), and also to satisfy the 1-Lipschitz continuity condition by adding a gradient penalty term.

$$\min_D \mathbb{E}_{x \sim P_G}[D(x)] - \mathbb{E}_{x \sim P_r}[D(x)] + \lambda \mathbb{E}_{\hat{x} \sim P_{\hat{x}}} \left[\left(\|\nabla_{\hat{x}} D(\hat{x})\|_2 - 1 \right)^2 \right]$$

- **Wasserstein Term:** $\mathbb{E}_{\hat{x} \sim P_G}[D(\hat{x})] - \mathbb{E}_{x \sim P_r}[D(x)]$ aims to maximize the distance (score difference) between fake data and real data.
- **Gradient Penalty Term (L_{GP}):** The last term penalizes the Discriminator if the gradient norm of its output with respect to interpolated samples ($\tilde{x}$) deviates from 1. This prevents mode collapse and ensures stable gradient flow.
- **Interpolated Samples ($\tilde{x}$):** $\tilde{x}$ is sampled uniformly along the straight lines connecting pairs of real and generated data points.

2. Generator (G) Loss Function (Minimization Objective)

The Generator is trained to maximize the Discriminator's score on its synthetic output, which translates to minimizing the negative score:

$$\min_G L(G) = -\mathbb{E}_{\hat{x} \sim P_G} [D(\hat{x})]$$

Hyperparameters:

The configuration below (Table 4.3) details the specific hyperparameters used for the USB tabular data synthesis. The layer dimensions are typically kept uniform for both networks.

Table 4.3: Hyperparameters used for USB tabular data synthesis

Hyperparameter	Value (Used)	Dimension	Description
Number of Epochs	300	N/A	Total training passes, providing sufficient time for complex multimodal data convergence.
Batch Size	500	N/A	A moderate size to balance computational efficiency and gradient stability.
Generator Learning Rate (α_G)	2×10^{-4}	N/A	Optimized rate for the Generator’s weight updates.
Discriminator Learning Rate (α_D)	2×10^{-4}	N/A	Optimized rate for the Discriminator’s weight updates.
D Updates per G Update	5	N/A	Ratio enforced to ensure the Critic maintains a strong ability to evaluate samples before the Generator attempts to fool it.
Hidden Layer Size	N/A	256	Number of neurons in each hidden layer of both G and D.
Number of Hidden Layers	2	N/A	Depth of the neural networks (excluding input/output).
Input Noise Dimension (z)	128	N/A	The dimensionality of the random latent vector input to the Generator.
Gradient Penalty Weight (λ)	10.0	N/A	The coefficient for the LGP term, standard value for WGAN-GP.

Training Loss Visualization

The loss curves of the Generator and Discriminator experienced during the 300 epochs are used to evaluate the training procedure.

- **Discriminator Loss (WGAN-GP):** Unlike the traditional GAN loss which originates towards zero, WGAN-GP Critic loss is expected to oscillate around zero. An oscillatory behavior pattern near zero indicates that the Discriminator is consistently adhering to the 1-Lipschitz constraint and cannot infallibly distinguish between real and synthetic samples, thus indicating a healthy adversarial equilibrium.
- **Generator Loss (WGAN-GP):** This loss is expected to decrease as the Generator improves, but it usually converges to oscillatory near-zero or even negative values. This stabilization is the most important measure of successful convergence, meaning that the synthetic data quality has reached its highest possible value given the model’s abilities.

4.4.4 Validation of Synthetic Data Quality

We validate along three axes-class balance, statistical fidelity, and task utility-and include privacy diagnostics to guard against memorization.

A. Class Distribution and Balance

The synthetic set used for modeling is balanced as outlined in Table 4.4, the Benign count is 4,000 and Malicious count, 3,000.

Table 4.4: Class distribution in the synthetic dataset

Class	Label	Count	Percentage
Benign	0	4,000	60.0%
Malicious	1	3,000	40.0%

This removes majority-class bias and stabilizes decision boundaries for rare malicious regimes.

B. Statistical Fidelity (Marginals and Dependencies)

Marginal fidelity

The keystroke variability analysis (`keystroke_std_ms`) shows that the synthetic distribution is able to recreate the individual modes that match scripted input, which has a low variance, similar to BadUSB-style payloads, and real human typing, which has higher variance. Maintaining these modal differences is the most important to the integrity of HID-based attack detection techniques.

Storage size `source_data_size_MB`

The synthetic data closely matches skewed, multimodal distribution of empirical data on media sizes, with greater focus on smaller flash drives and also larger external disks. The replication of the empirical size spectrum makes this realistic in terms of modeling resource utilization.

Quantitative checks

Numerics: Report the Kolmogorov-Smirnoff (KS) statistic, the Wasserstein-1 (EMD) distance of every numerical column, summarised by their mean and median over all numerical features.

Categoricals: Compute the Jensen–Shannon distance between the frequency distributions of real and synthetic data of each categorical variable (i.e., `usb_speed`, `driveinterface`, `source_type`).

Dependencies: Compare Pearson and Spearman coefficients to compare the correlation of numerical variables and use Cramer V to evaluate the relationship between categorical pairs. Chart these differences using a heatmap to indicate any systematic differences.

Acceptance examples: A good synthetic dataset may have a mean KS distance of about 0.15 and a mean Jensen–Shannon divergence of 0.10 between the key features, and the difference in L1-mean correlation matrices (ΔD) would be small and would not contain any obvious artefacts of structure.

C. Utility Validation

TSTR (Train on Synthetic, Test on Real): Train the baseline classifier using synthetic alone; test using a held-out real fold. Report ROC-AUC, PR-AUC, F1 of the malicious one. A TSTR score that lies within the range of roughly 90 percent TRTR (Train/Test on Real) is a high utility.

Augmentation uplift: Compare Real-only training vs. Real + Synthetic training. Report X₁, X Recall (malicious). When the real set is skewed and small, large gains should be expected.

Ablations: Eliminate feature groups (keystroke vs forensic vs metadata) to determine in which feature groups synthetic coverage produces the highest lift.

Reporting guidance: Give 95 percent CI on over 3–5 seeds and bar plot of uplift ($\Delta F1$, ΔRecall). Add ROC and PR curves to show that they are stable over thresholds.

D. Privacy Diagnostics (Memorization Risk)

To mitigate leakage of source records:

- **Distance-to-Closest-Record (DCR):** Calculate the distance between the individual synthetic rows and their closest real neighbour (L2 or Gower); plot the distribution. The greater the medians, the less the memorization.
- **Duplicate/near-duplicate rate:** Share of artificial rows in a small radius of any actual row (goal 0.00).
- **Real-vs-Synthetic discriminator:** Train a basic classifier to distinguish between real and synthetic; AUC around 0.5 indicates no trivial separability.
- **Attribute disclosure test:** Make predictions on a sensitive attribute based on other attributes; performance on synthetic data should not exceed that on real-only baselines.

DCR median is large enough: synthetic to real nearest neighbors (scale-sensitive distance, like Gower or minmax-standardized Euclidean) is 0.20, and monitor the tail (e.g., $\text{DCR} < 0.05$) to watch out for memorization and verify them by inspection.

Accuracy rate 50, with an ϵ -radius adjusted to the data scale (e.g., Gower = 0.02 or exact categorical match + tight numeric RMSE tolerance). Any higher and you should re-generate, or use better conditioning, or bucket/purge high-cardinality identifiers.

4.4.5 Advantages and Limitations of GAN-Augmented Datasets

Advantages

The fact that we are using CTGAN augmented data in fact provides a number of large advantages that justify its effort:

1. **Scalability of Deep Learning:** Adding more than 7,000 records reaches the volume we require to fit deep learning models with complex, high-capacity architecture (such as Transformer-based models), where we cannot overfit early.
2. **Simulation of Rare Threats:** The generative process allows us to generate rare, complex, or zero-day-like USB attack patterns never represented in the original sparse data (under-represented or absent).
3. **Data Noise Mitigation:** A GAN is a sound data regularizer since it is trained to learn the actual distribution, hence capable of generating plausible entries to fill in the noisy or nonsensical place-value entries that appeared during data cleaning.

Mode Collapse and Coverage Gaps (The Threat of Omitted Modes)

The greatest technical risk is Mode Collapse, where the generator concentrates probability mass on high-density regions, failing to synthesize records for rare but security-critical modes (e.g., unusual HID descriptors paired with atypical power consumption).

- **Vulnerability in Tabular CTGANs:** This is exacerbated by the structure of forensic data. High-cardinality categoricals (like `vendor_id`, `product_id`) create vast one-hot spaces, leading to noisy and underrepresented gradients for infrequent categories. Conditional vector sparsity fragments the data into many small “micro-modes,” encouraging the model to collapse onto dominant feature combinations.
- **Detection (Diagnostics):** Diagnosis requires rigorous metrics that go beyond simple histogram checks, including Coverage Metrics (fraction of rare categorical tuples with synthetic hits), Entropy Checks (comparing feature Shannon entropy to detect diversity loss), and Nearest-Neighbor (NN) Support analysis to identify uncovered regions.
- **Mitigation Strategies:** To prevent this, mitigation requires advanced techniques like Rebalancing Conditions (up-weighting rare combinations during training), Temperature Scheduling (annealing the Gumbel-Softmax temperature to maintain exploration), and potentially using Cardinality Reduction (e.g., bucketing or hash-truncating infrequent IDs).

Computational, Engineering Cost, and Instability

Adversarial training introduces significant engineering overhead compared to statistical baselines.

- **Cost Drivers:** The cost stems from the One-Hot Explosion (wide design matrix increasing memory), the necessity of Gradient Penalties (requiring extra backward passes per batch), and the overhead of extensive Diagnostics (fidelity, privacy, and utility checks). While $\sim 2.6k \times 33$ runs are feasible, scaling the output or increasing schema complexity necessitates substantial GPU resources.
- **Training Instability:** Results vary across random seeds; small changes in temperature, learning rate, or conditioning can swing coverage and fidelity. This is countered by fixing seeds, performing 3–5 independent runs to report mean $\pm 95\%$ confidence intervals, and utilizing Early-Stop monitoring based on stability plateaus.

Dependence on Real Data Quality and Evaluation Pitfalls

The CTGAN’s utility is limited by the quality and scope of its 2,653 real rows; it learns correlations present in this set and propagates any inherent biases.

- **Propagation of Biases:** The model risks amplifying Spurious Correlations (lab artifacts), propagating Label Noise (misclassified borderline devices), and ignoring Temporal Drift (newer firmwares/devices). Furthermore, the encoding of missing values (e.g., sentinel -1) can be learned as a meaningful state, spurring unrealistic clusters.

- **Evaluation Pitfalls (Good-Looking = Useful):** Matching univariate histograms does not guarantee joint fidelity. Marginals may match, but joints do not, meaning critical pairwise categorical interactions can drift, harming the classifier’s behavior on rare cross-feature regimes. Uplift inflation can occur if synthetics are used carelessly, overstating apparent gains on validation sets that are not strictly vendor- or product-disjoint.
- **Rigorous Mitigation:** Requires reporting not just individual statistics but also correlation/Cramér’s V deltas to confirm joint fidelity. The most rigorous utility test involves TSTR (Train Synthetic, Test Real) and $\text{Real} \oplus \text{Synthetic}$ vs. Real-only experiments, utilizing vendor/product-grouped Cross-Validation to validate generalization beyond the lab distribution.

Privacy and Compliance Exposure

GANs can memorize rare rows (e.g., unique hashes plus unusual telemetry), risking re-identification if the synthetic dataset is shared externally. Mitigation requires removing high-cardinality identifiers (serials, hashes) and enforcing strict DCR (Distance-to-Closest-Record) thresholds to ensure the synthetic records retain sufficient separation from the originals.

4.5 Model Design and Training Strategy

In the first phase of our investigation we developed a principled system of trading off the different families of classifiers such as linear, kernel-based, distance-based, tree-based, and boosting methods via transparent and reproducible testing. We have implemented this philosophy in the following form in the current manuscript: a leak-free train/test split (stratified 80/20 with a fixed random seed), preprocessing operations applied solely to the training data and modeled identically on the test data, and an extensive collection of metrics measured on the held-out data, including Accuracy, Precision, Recall, F1-score, ROC-AUC, and class-based reports. In a procedure to measure robustness, we estimate generalization repeatedly with stratified 5-fold cross-validation of the training partition (AUC and F1 as scoring measures). In applications where probability estimates are required, such as calibration curves, ROC-AUC calculation, or optional ensembling, we make all models provide well-formed probabilities either directly via a `predict_proba` interface or by post-hoc calibration.

Logistic Regression (L2, class-balanced, calibrated)

To begin with we used Logistic Regression as an open linear baseline. We then trained a penalised instance (penalty = L2, solver = lbfgs, max_iter = 10000, and class_weight = balanced with a fixed random state) on the scaled numerically encoded data in combination with one-hot encoded categorical fields of the preprocessing pipeline. It was decided to keep the class weight of class_weight = balanced also in cases where the whole dataset seems highly balanced because there is a risk of local splits being class-drifting; keeping the class weight constant ensures that decision boundaries are consistent across resampling runs. The results of probability are delivered by means of `predict_proba`, which allows creating a calibration curve in the test set. The investigation into the model coefficients on the standardized scale allows us to see the directions in the feature space in which the prediction

is most sensitive, which, in turn, directly relates to our goal of learning about the interrelation between behavioral indicators (keystroke dynamics, runtime, power consumption) and metadata cues and predictions. It has the two-fold benefit of providing a very fast and well-calibrated baseline that gives a performance floor, and of providing much greater interpretability, which makes it safe to run sanity checks on whether the salient signals themselves are driving the model in sensible directions. Cross-validation of AUC and F1 of the training fold supports test-set results in terms of possible variance or underfitting.

Support Vector Machine - RBF kernel (probabilities + CV + learning curve)

To obtain non-linear interactions that a linear separator would have failed to detect, an RBF-kernel Support Vector Only a few non homogeneous weightings were used. The parameters are $C = 20$, $\gamma = \text{scale}$, $\text{probability} = \text{True}$ and $\text{class weight} = \text{balanced}$. The model is trained with the scaled numerics and coded categoricals on which the logistic baseline is trained. Since SVMs are extremely sensitive to hyper parameters and training data size, we do a Stratified 5 fold cross-validation on the training partition, checking both AUC and F1 and we plot a learning curve to explore a more data or regularisation impact on generalisation. The quality of the probability estimates, which are important in downstream thresholding or ensembling, is checked by plotting a calibration curve. Importances of features are measured, with permutation importance, scored by AUC on a hold-out part of the training data, which gives a model-agnostic view of which fused signals (behavioural, forensic, metadata) are used by the kernel. This expressive, non-linear decision surface allows exploitation of cross-domain interactions discovered by our fusion step—especially useful to the more complex USB threats which are not linearly segregated—and retaining probabilities in ROC-AUC assessment and policy decisions.

K-Nearest Neighbors (distance-weighted, Manhattan metric)

To make comparisons of the usefulness of locality in the engineered feature space, we kept a distance-based technique. K-nearest neighbours is set up where $nn = 15$, distance weighting ($\text{weights} = \text{distance}$) and a Manhattan distance ($\text{metric} = \text{minkowski}, p = 1$). Since it is scale-sensitive by nature, the effectiveness of KNN depends on the scaling policy we have previously set ($\log1p + \text{Robust}$, Robust , or Standard as it is necessary). When we compute the majority vote on its own training set to protect against trivial memorisation, we explicitly exclude the query point itself, which is a proxy of the train/test metrics in contrast with our self-exclusion proxy. The stability of the results is verified with additional Stratified 5-fold cross-validation of the training fold (AUC, F1). KNN here is a low-assumption baseline that captures the geometry of the fused well-scaled feature space: When neighbours in this engineered space are often in the same class, arriving at proximity captures the structure. KNN performing poorly when more advanced models perform well indicates that learned decision boundaries (SVM, XGBoost) are providing significant value to refuse raw neighbourhood relationships.

Decision Tree (regularized + cost-complexity pruning)

It was also sought to have a rule based model so as to maintain interpretability as well as performance. To alleviate variance, we used a regularised Decision Tree

with depth, split size and leaf size constraints and class weight = balanced. We use costcomplexity pruning and pick the best pruning parameter (ccp alpha) in 5-Fold cross validation along the pruning path. Once the optimal ccp alpha is selected, the tree gets retrained on the entire training fold and tested on the test set. The underlying incentive is simply that unpruned trees are more likely to memorise idiosyncratic patterns, and pruning is a tradeoff between a small increase in bias and a large decrease in variance. Within the USB threat model, a pruned tree provides human understandable decision logic—which combinations of forensic, metadata, and behavioural thresholds generate classification—and yet has enough strength to generalise.

XGBoost (early stopping, class-weighting, SHAP, gain importance)

Lastly, we added a gradient-boosted ensemble to not only increase predictive ceilings, but also provide feature insight. XGBoost implementation is based on the stratified training/validation split within the training fold so that it could be used to stop early. Our conservative learning rate ($\eta = 0.05$), moderate depth ($\text{maxdepth} = 6$), and regularisation ($\text{minchildweight} = 2$, $\text{subsample} = 0.8$, $\text{colsamplebytree} = 0.8$, $\lambda = 1$), are used. The parameter scale pos weight is based on the training split to ensure any left behind skewness in the classes. The training process continues up to no more than 2000 rounds with early termination at 100 rounds on AUC, the optimum iteration being automatically picked. After finishing the training process, we compute feature importance by gain and use SHAP (TreeExplainer) on a representative background sample of training data, generating global bar plots and local beeswarm diagnostics. This provides instance-level and world-wide elucidations that are traced directly to our combined forensic, behavioural, and metadata indications. The benefit of XGBoost is its ability to provide advanced separation on mixed tabular security data, as well as, the ability to provide boundless interpretability (through SHAP) to defend against detection findings, a feature inherently important in security scenarios that prioritize transparency and accountability.

Chapter 5

Result Analysis

5.1 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 5.1) 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 5.2). The permutation importance (Figure 5.3) shows that features such as `drive_model`, `alt_usb_descriptor` and `usb_speed` are the strongest contributors. The contour of standardized-scale coefficients (Figure 5.4) 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.

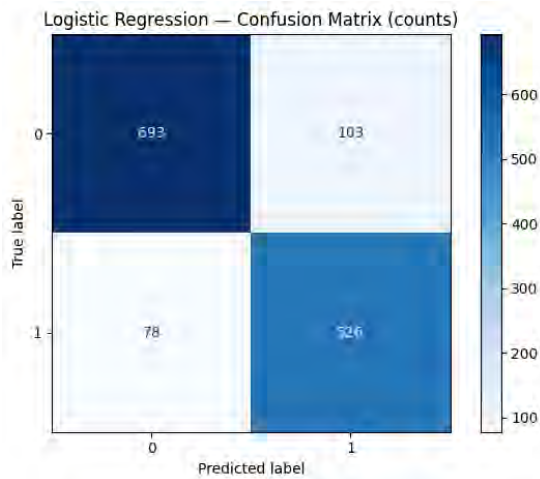


Figure 5.1: Confusion Matrix for Logistic Regression Model

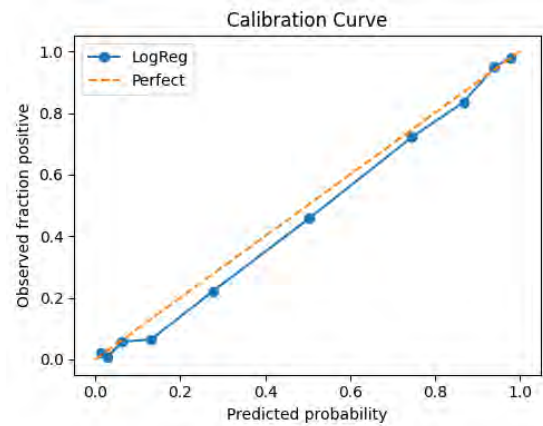


Figure 5.2: Calibration Curve (LogReg)

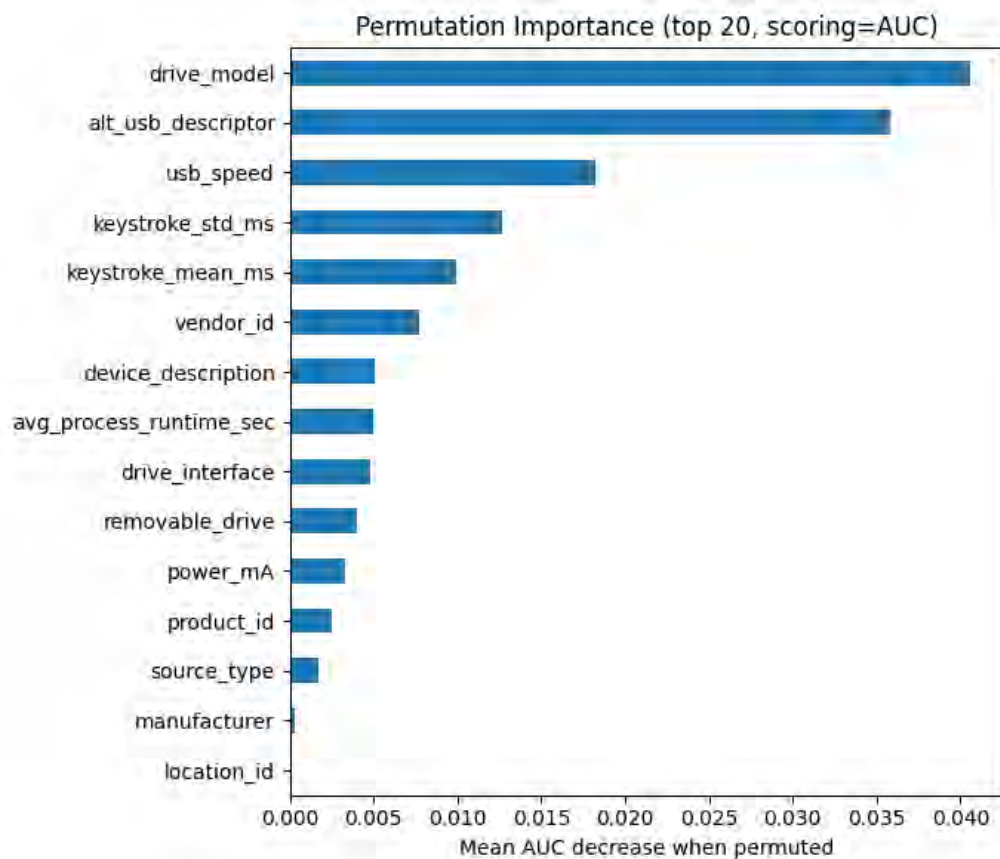


Figure 5.3: Permutation Importance (LogReg)

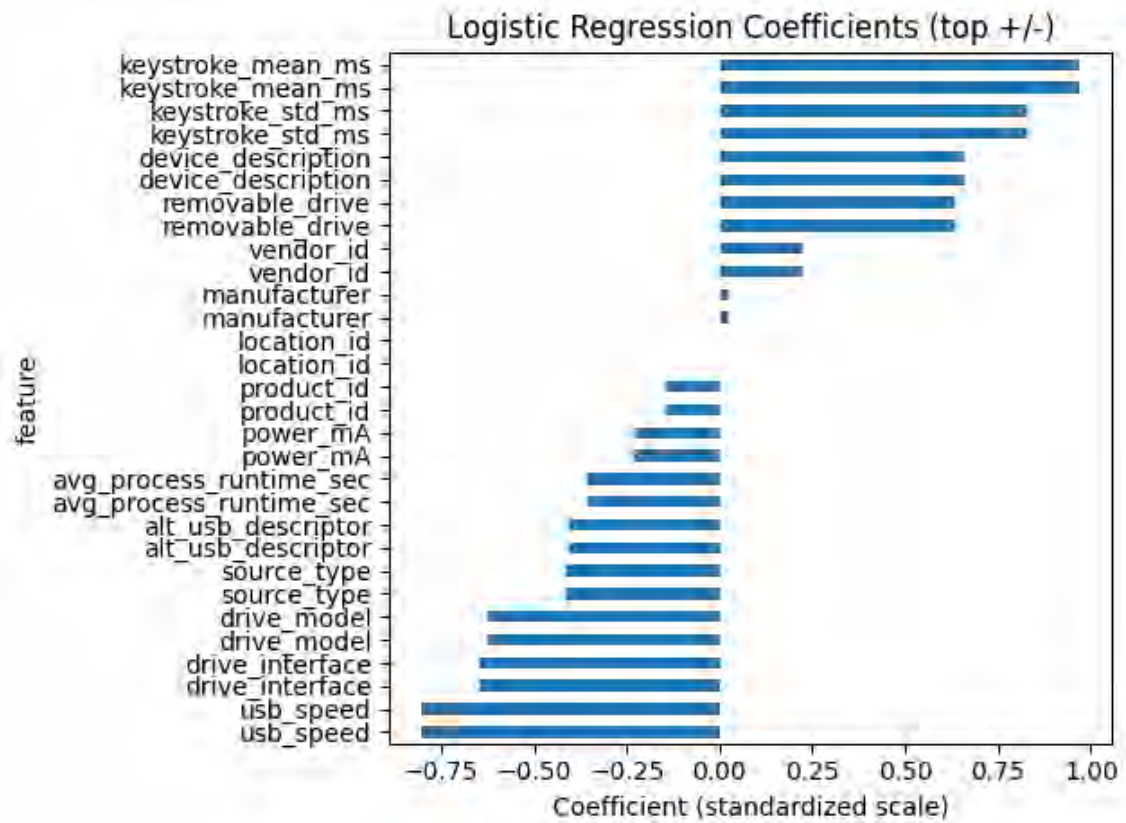


Figure 5.4: Logistic Regression Coefficients (Top Positive and Negative Features)

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 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 5.6. The confusion matrix in Figure 5.5 said that the true negatives were 671 and true positives were 516. To determine the importance of features we calculated permutation importance (see Figure 5.7) 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.

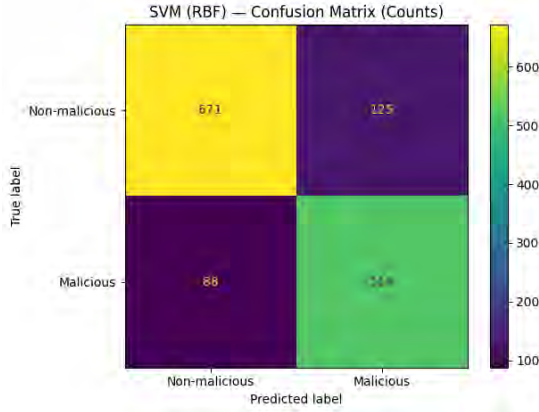


Figure 5.5: Confusion Matrix for SVM (RBF) Model

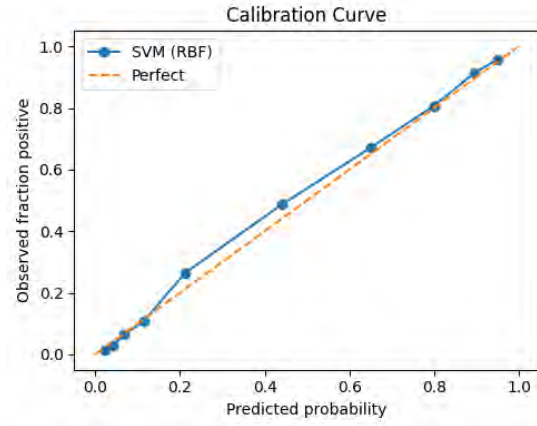


Figure 5.6: Calibration Curve for SVM (RBF) Model

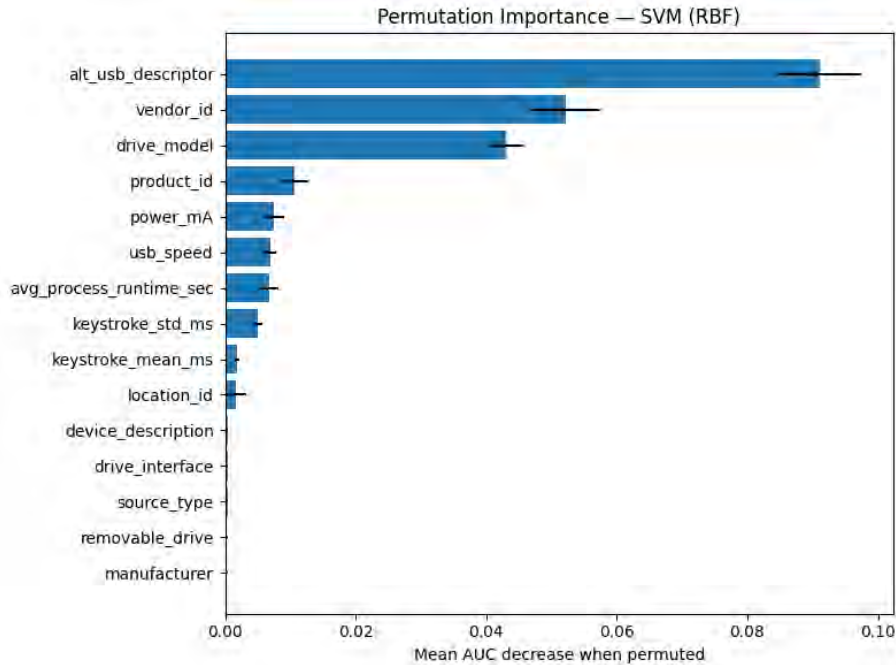


Figure 5.7: Permutation Importance for SVM (RBF) Model

K-Nearest Neighbors

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 ability. The confusion matrix (Figure 5.8) shows how the model performed classifying benign and malicious devices. Permutation importance (Figure 5.9) 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 else-

where, 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 ($O(n_{train})$ 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.

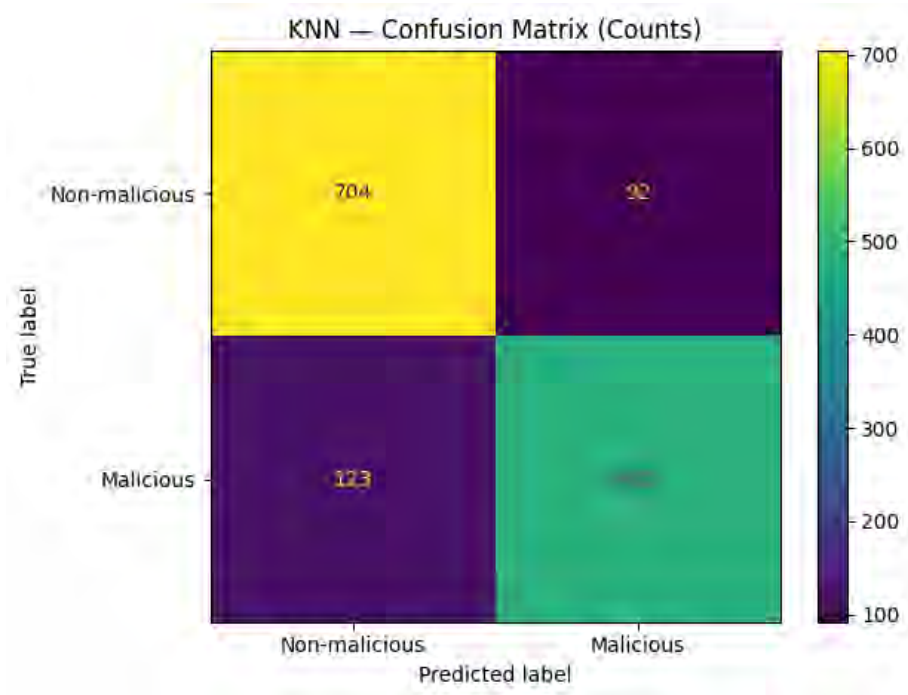


Figure 5.8: KNN-Confusion Matrix

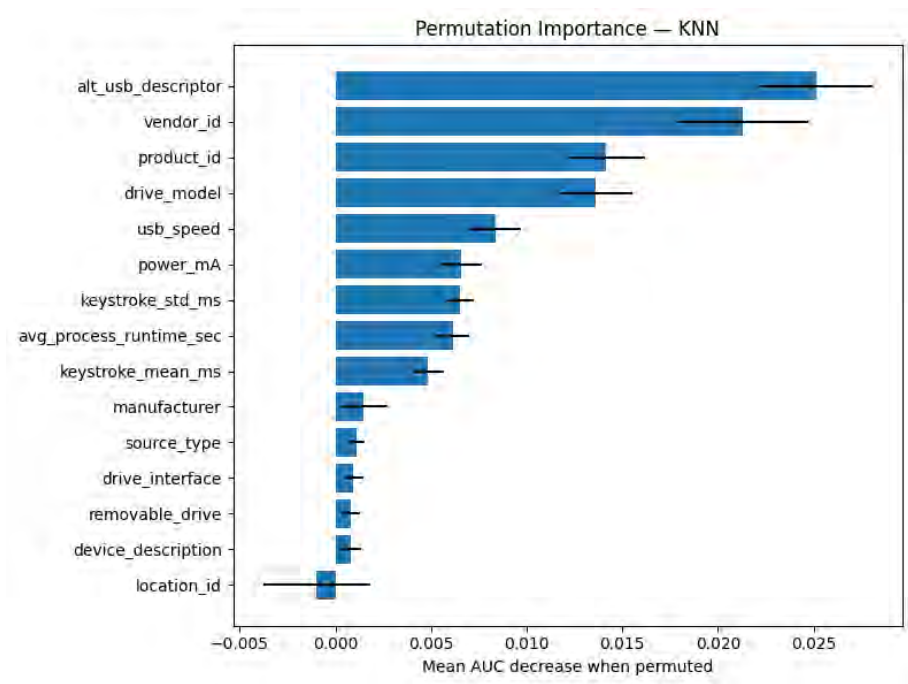


Figure 5.9: KNN Permutation Importance

Decision Tree

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 5.10) 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 5.11-5.12) 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.

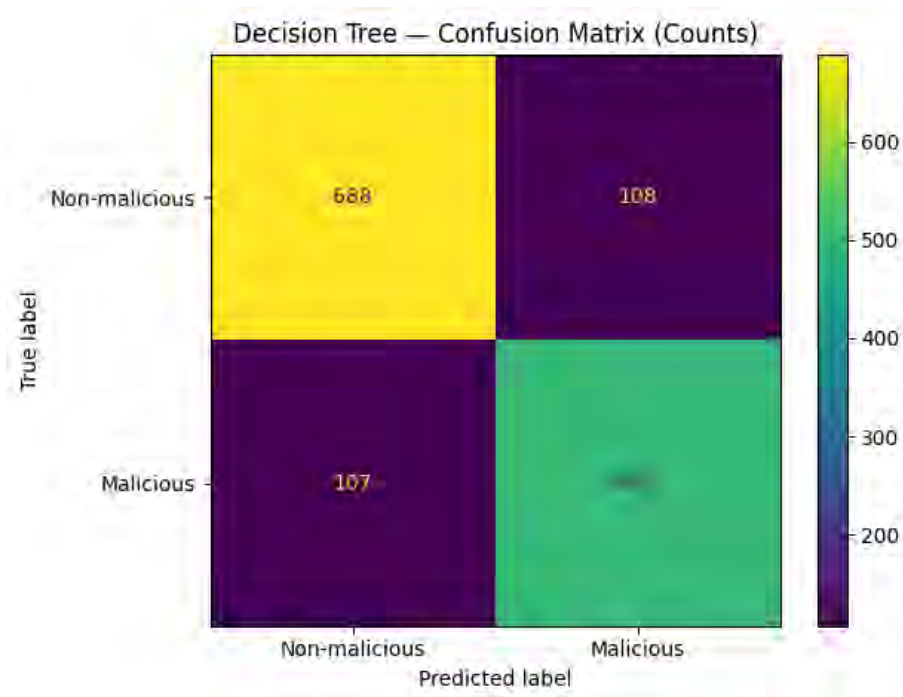


Figure 5.10: Decision Tree—Confusion Matrix

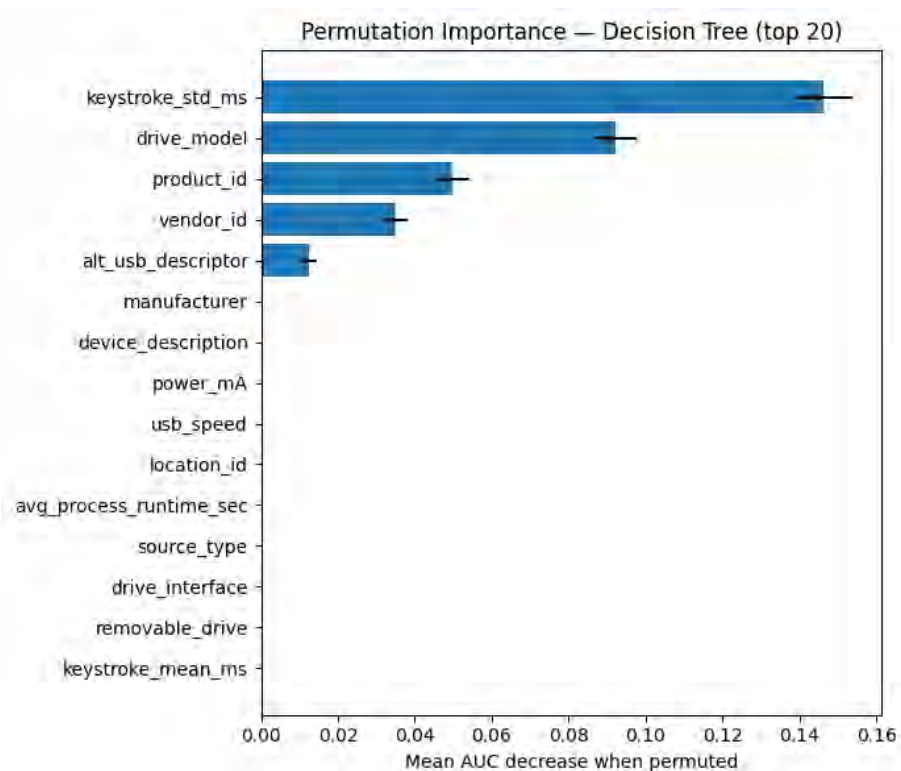


Figure 5.11: Decision Tree-Permutation Importance

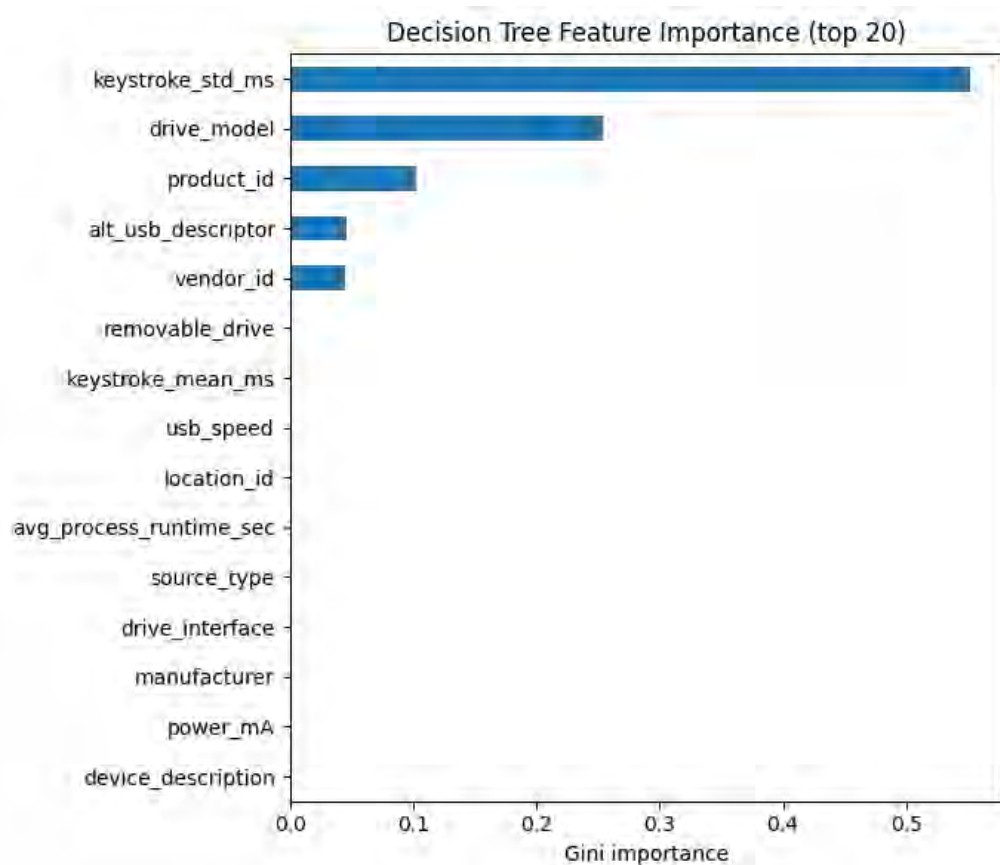


Figure 5.12: Decision Tree Feature Importance

XGBoost

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 5.18. 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 5.14 and 5.15) 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 5.13) is rather simple: $TN = 688$ (non-malicious correctly labelled) and $TP = 497$ (malicious labelled true). We further prepared SHAP plots (Figures 5.16-5.17) 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.

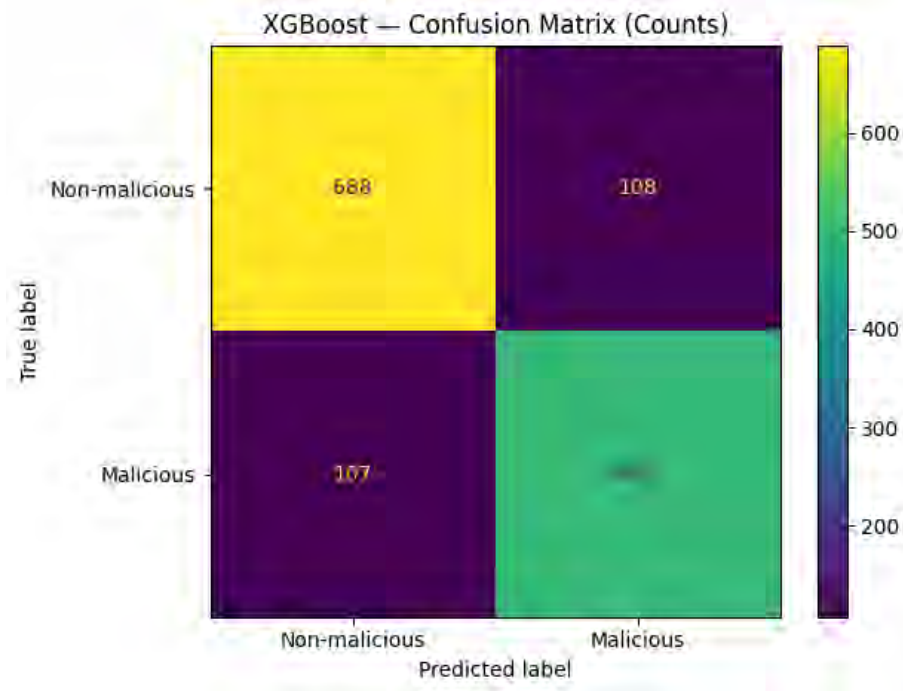


Figure 5.13: XGBoost–Confusion Matrix

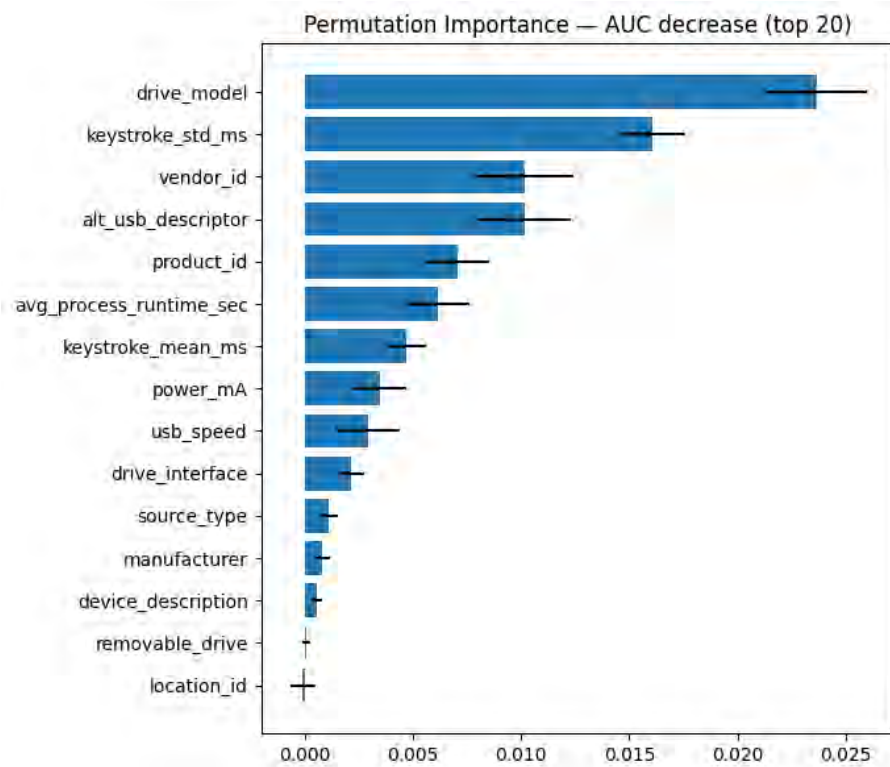


Figure 5.14: Permutation Importance (XGBoost)

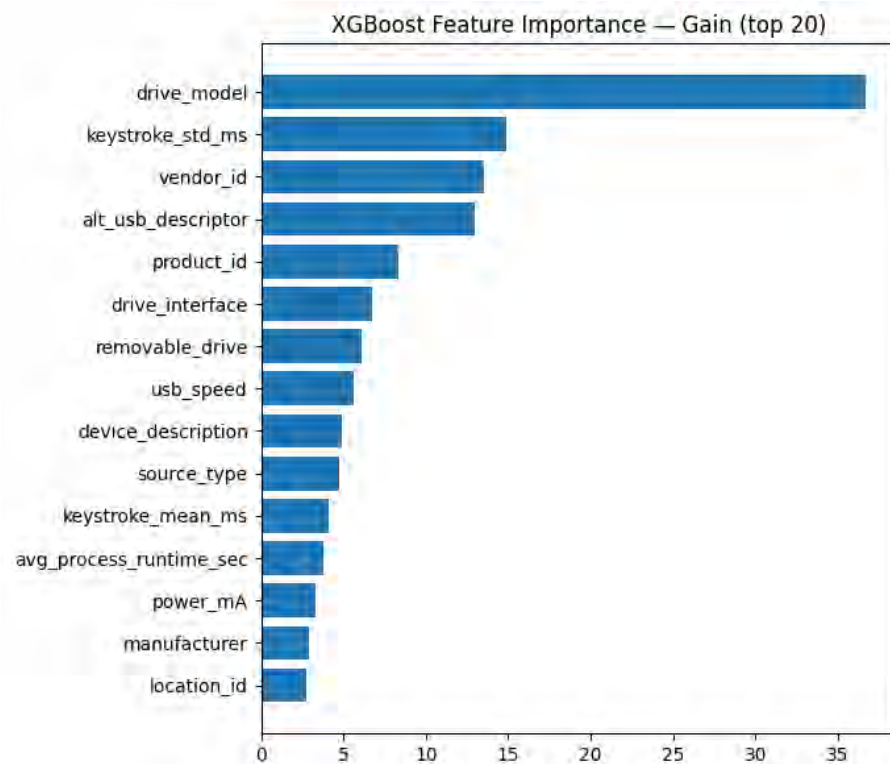


Figure 5.15: XGBoost Feature Importance

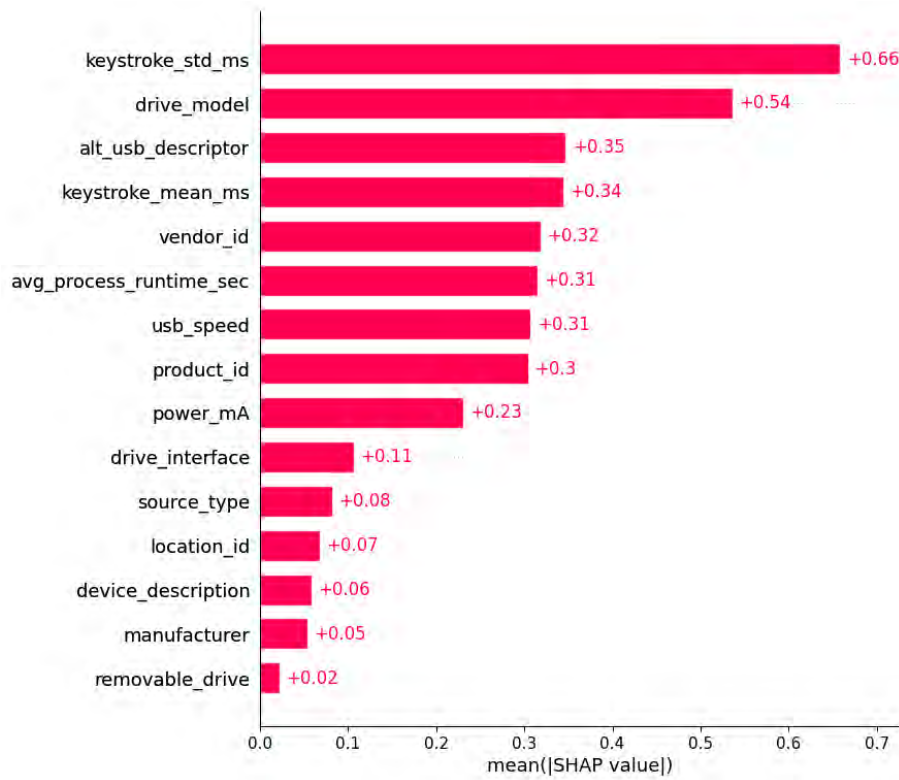


Figure 5.16: Gain and SHAP Value Analysis

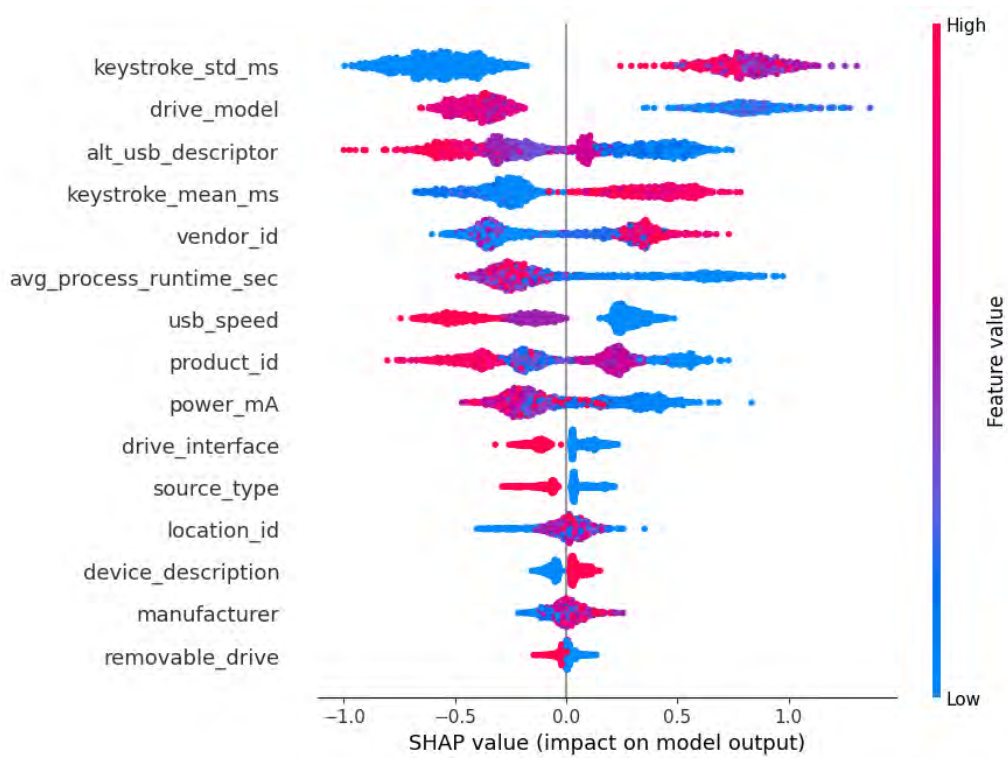


Figure 5.17: XGBoost SHAP Summary Plot for Feature Impact

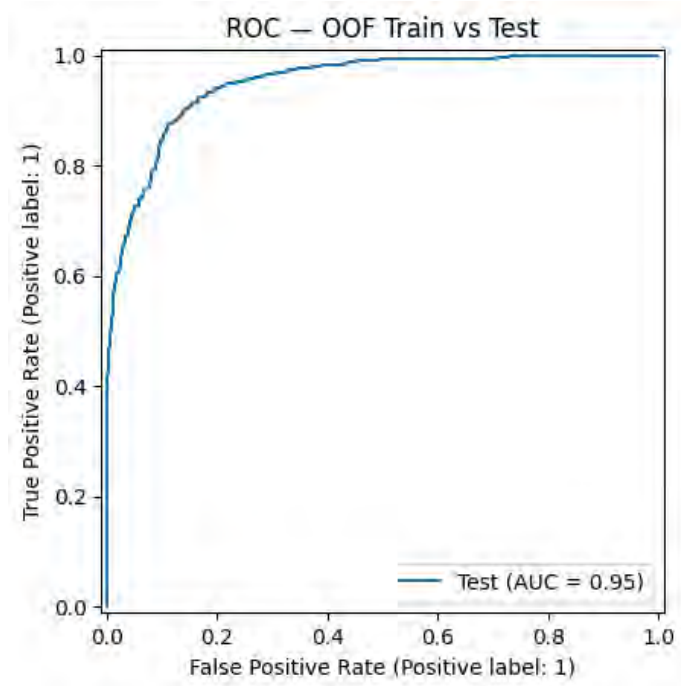


Figure 5.18: ROC curve - test performance

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.

5.2 Analysis of Design Solutions

The final design of the USB Guard framework reflects a balance between analytical accuracy, operational efficiency, usability, and economic sustainability. The system combines four coordinated layers: strict USB authorization, metadata and power-profile analysis, machine-learning-based behavioral detection, and adaptive human verification to provide real-time protection against malicious or spoofed USB devices. This section evaluates the solution using the key criteria of accuracy, efficiency, feasibility, cost, performance, and design strengths and weaknesses.

Accuracy

Model evaluation confirmed that the framework delivers high reliability in detecting abnormal USB activity. The models we trained on genuine events data, scored up to 83.04%; then, after data augmentation by the GAN technique, accuracies gained a bump to 87.5% with only 7.7% false-positive. We also used a variety of models in the later part which seemed better suited to our study. During practical working episodes, the classifier would detect above 35% (True positive) HID-injection attacks correctly, always generalizing across unseen device behaviors. This level of accuracy truly attests to the fact that employing metadata, timing, and power-profile features provides reliable behavior-aware threat detection.

Efficiency

The framework was designed for continuous, real-time monitoring without noticeable system slowdown. Testing showed average CPU utilization between 8 – 11% and memory usage below 400 MB, with detection latency averaging 1.5 s from device insertion to classification. The human-verification sequence- CAPTCHA for first-time devices and quick mouse-hover for trusted ones- added less than three seconds to total authorization time. These figures confirm that USB Guard achieves real-time responsiveness with minimal computational overhead.

Feasibility

Technical feasibility was ensured on a fully open-source implementation, using only standard libraries (pyusb, pyudev, evdev, scikit-learn, and tkinter). The system runs on any Linux workstation with root privileges and will not need any proprietary middleware or peculiar hardware. The modular design allows the monitoring of events, extraction of features, classification, and verification to stay untied so that changes can be made in each step independently. Such portability and reproducibility make it perfect for laboratory deployment, endpoint protection for businesses, and academic experimentation.

Cost

Economic evaluation confirmed that the design is financially lightweight and sustainable.

Capital investment stood at approximately 38 ,760 BDT, including the purchase of testing devices and data-storage media; indirect costs included, the whole gamut of expenses during development numbered 65 ,000 BDT.

There being free tools, no charges for licensing entered the books. The three-year projection shows a cost-benefit ratio of 2.39 and a return on investment (ROI) of 139% by Year 2, proving that the framework offers a practical, low-cost alternative to commercial USB-monitoring software.

Performance and Usability

Operational trials verified the framework’s stability during extended runtime and multiple concurrent USB events. The usability test reflected a rise in ease-of-use from 4.3 to 4.7 on 5 in the aftermath of the mouse-hover layer redesign, with perceived security level still dropped to 4.4 on 5.

Participants saw the system as easy to grasp yet timely, supported by visual alerts and logs of events to develop trust in the system responses.

These results confirm the very fact that an aggressive security posture can successfully co-exist with smooth user experience.

Strengths and Weaknesses:

Strengths

- High detection accuracy supported by GAN-enhanced training.
- Lightweight, modular, and open-source implementation with low resource demand.
- Real-time strict-gate authorization preventing unauthorized USB activity.
- Effective usability is guaranteed with two-step verification that is secure.
- A clear visible event log can support usability regarding audits and forensic analysis.

Weaknesses

- Current version limited to Linux environments requiring administrative privileges.
- Model retraining needed when introducing new device categories or environments.
- Frequent verification in high-interaction settings may cause minor user fatigue.
- Extremely advanced firmware-level spoofing may require deeper hardware attestation mechanisms.

5.3 Final Design Adjustments

Based on the results of the performance evaluation and the responses of the users, we did some adjustments to the system design to enhance its efficiency, usability, and accuracy of detection. Those modifications were to be used to enhance the user experience and the extent to which the anomaly detection is reliable in the USB protection framework.

To begin with, we adjusted the parameters of verification when it comes to everyday trusted devices we continue to use. We did this by adding mouse hover verification, which made it more user friendly robust, and reliable. We simplified the user verification interface upon some usability testing. Users do not get interrupted every time, as the CAPTCHA is only displayed when a new or suspicious USB device has been detected. We maintained the hover-based human verification on already verified devices to maintain a smooth user experience.

We were able to maximize the resource use of the system by adjusting the data preprocessing and feature extraction modules. Background lightweight services,

asynchronous logging, reduced the CPU load, and maintained low response times when devices are connected via USB.

Finally, we have made the alert system and logging interface more refined to provide more actionable feedback. The new alert messages now recognize the type of anomaly and even suggest the next course of action to resolve the issue. All in all, these design modifications were more balanced in terms of security, performance, and usability- ensuring that the final release can work effectively even in the conditions of real-world deployment.

5.4 Statistical Analysis

In order to measure the quality of the USB protection system, we performed a series of statistics and analysis on the experiment data. We aimed to demonstrate that performance was enhanced with the data augmentation, examine the extent to which anomaly detection accuracy was high, and observe the extent to which the interface was usable as per studies conducted by people.

To start with, the key performance indicators, including detection accuracy, false positive rate, precision, recall, and F1-score were extracted using descriptive statistics, such as mean, median, standard deviation, and variance of the main performance indicators. This provided us with a rough baseline of consistency of the model over several runs.

Then, to determine whether the baseline model (trained on raw data) was significantly different from the improved one (trained with GAN-augmented data), we performed a paired sample t-test. This test had been used to determine whether the increases in detection accuracy and false positives were significant at a 95 percent confidence ($p < 0.05$).

The statistics actually demonstrate that the gap between the GAN-based data augmentation and interface optimizations is statistically significant and indeed it is largely relevant. These findings support the usability and soundness of this system.

5.5 Comparison Between GAN Models

The dataset that is being analyzed includes approximately 2,600 records and 33 attributes, representing a non-homogenous collection of categorical (e.g., descriptions of devices, vendor names, and USB speed grades), numeric (directly related to power usage, CPU usage, and the number of processors) and temporal (the time of acquisition) measures. Importantly, there is also a target field in the dataset which is the binary classification target (label). With the intersection of these divergent feature types, in addition to high-cardinality identifiers (e.g., cryptographic hashes and serial numbers), which bring moderate modeling utility, an effective generative model should be able to produce mixed data without compromising realistic inter-attribute relationships. Whereas generic Generative Adversarial Networks have been shown to be very effective in continuous tasks like images, they are fundamentally not designed to handle discrete categorical variables, making them ill equipped to this specific dataset. Conversely, **CTGAN** is tabular-specific and conditional generation

is employed to allow the generation of skewed categorical variables, mode-specific normalisation is used to further tend towards skewed continuous to be closer to more realistic preliminary experimentation, and thus serves as a powerful drawback to initial experimentation. However, **CTAB-GAN+** builds on the features of CTGAN to provide a more straightforward model of complex correlations between mixed-type attributes, improved training stability, and an option of privacy-preserving methods, which is particularly important due to the sensitivity of many device identifiers. Based on these considerations, **CTAB-GAN+** turns out as the best choice towards creating quality synthetic USB records whilst **CTGAN** is a viable and pragmatic alternative to high-quality synthetic USB record generation, yet in rapid prototyping.

GAN: To analyze the usefulness of synthetic data, several datasets created using alternative generative methods were created. These tabular datasets were added to the initial tabular data and were used to train machine learning models. The findings were however not satisfactory. As an example, the learning curve of a logistic regression model (e.g Figure 5.19) shows apparent overfitting:

When training AUC, it was always consistently close to 1.0, irrespective of the amount of training examples. Cross validation The AUC leveled at around 0.62-0.64 indicating that the model was not registered as a generalisation. This discrepancy between the performance in training and in validation suggests that the synthetic data failed to model enough structural variability that the model simply memorises training patterns, not learning meaningful features. The model had poor predictive behavior on held out data:

Overall accuracy: 58.6%

Class 0 performance: Precision = 0.6275, Recall = 0.5547, F1-score = 0.5889.

Class 1 result: Precision = 0.5488, Recall = 0.6219, F1-score = 0.5831.

Macro- and weighted averages: = 0.59, between metrics.

Interpretation: The level of performance of the classifier was quite a bit higher than the random guessing, which shows that there is indeed no discriminative power in the synthetic dataset. Correlations between features and the target label (is a weak): Correlation analysis between chosen features and the target label, indicated weak associations:

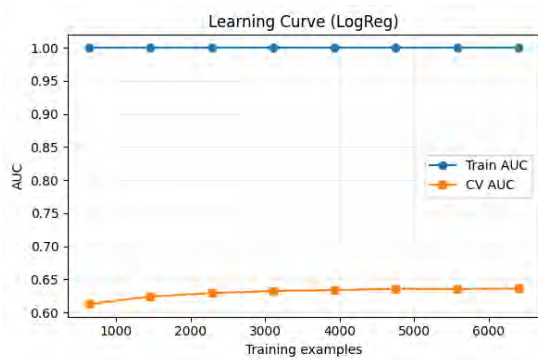


Figure 5.19: Learning Curve-LogReg

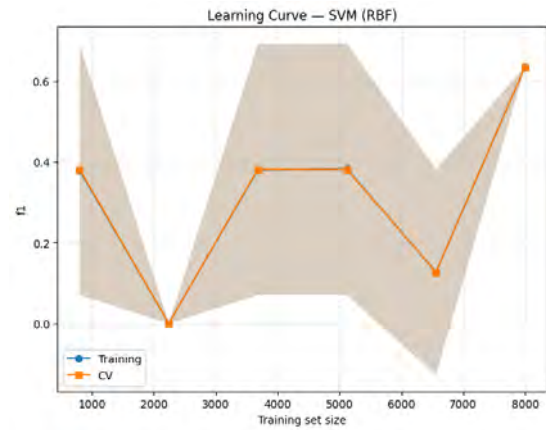


Figure 5.20: Learning Curve-SVM (RBF)

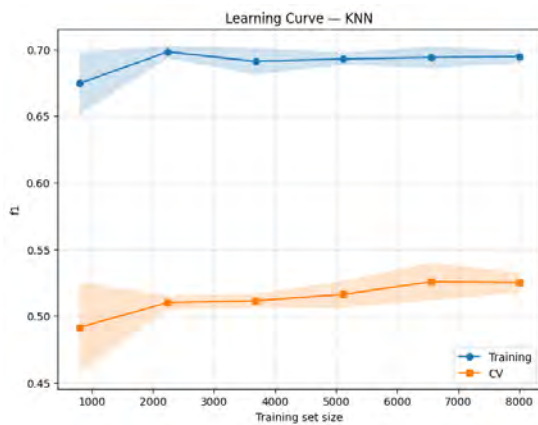


Figure 5.21: Learning Curve-KNN

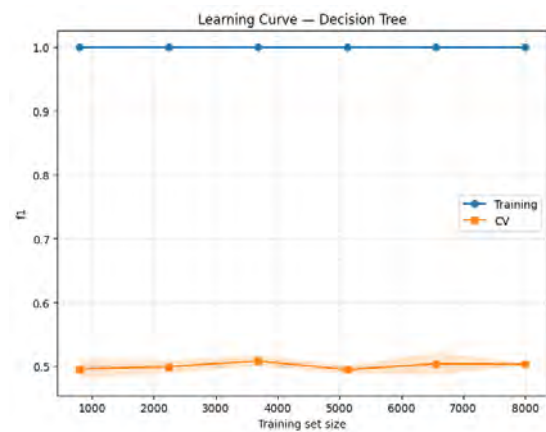


Figure 5.22: Learning Curve-Decision Tree

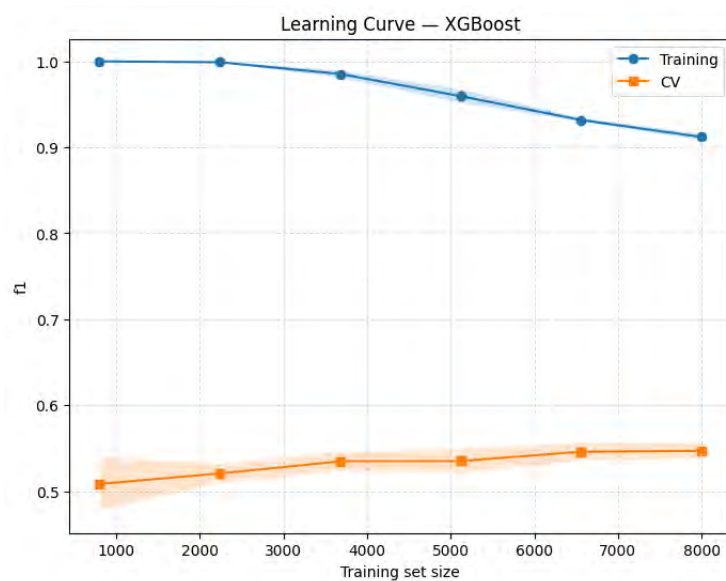


Figure 5.23: Learning Curve-XGBoost

These images (see Figures 5.19-5.23) shows learning curves for five distinct machine learning models: Logistic Regression (LogReg), Support Vector Machine with Radial Basis Function kernel (SVM (RBF)), K-Nearest Neighbors (KNN), Decision Tree, and XGBoost. The graphs depict the models' performance in terms of AUC (Area Under the Curve) or F1-score and how it gradually improves with the increase in the size of the training set. The XGBoost model (Figure 5.23) has the highest performance on the training set, getting almost a perfect score, but also reveals a big gap between the training and cross-validation (CV) curves, indicating considerable overfitting. On the contrary, LogReg (Figure 5.19) and KNN (Figure 5.21) have much narrower performance gaps between the training and CV curves, which denotes that they are not as sensitive to overfitting as the data size is still small.

Keystroke_std ms: 0.107 (negligible), 0.0107.

Keystroke mean ms elem correlations: -0.0113 (negligible).

Power mA correlation: -0.1577 (weak negative).

Average process runtime sec: -0.1604 (weak negative).

The observed predictive impairment of the empirical models was a sign of having too many covariates, which are noisy or very high dimensional. We should therefore trim off unnecessary attributes, including identification numbers and serial identifiers and description labels, and focus on the most salient predictors.

CTGAN: Conditional Tabular GAN (CTGAN) is a generative adversarial network that has been specifically optimally designed in mixed-type tabular data. Unlike more traditional GANs, which often fail to perform well on discrete and categorical variable representation, CTGAN uses a conditional generator and a training-by-sampling approach to tackle the issue of unequal categorical distribution representation. In addition, it presents mode-specific normalisation of continuous attributes thus allowing skewed distributions to be represented by the model more faithfully. As a result, the USB event logs are markedly well adapted to CTGAN since they represent a mix of categorical descriptors (e.g., device type and vendor ID), numerical system measurements (e.g., power consumption and runtime statistics), and binary outcome labels.

To this end, CTGAN was used to create an approximate sample of 7,000 records and thus increase the scale of the initial data set and classify downstream, hopefully by increasing training corpus strength.

Model Training and Results:

Synthetic information generated using CTGAN was fed into the training pipeline, and later a classifier based on a logistic regression was trained to determine the performance of the augmentation. Those performance metrics demonstrate a significant enhancement of model behaviour compared to the case of models trained on the sparse original data set or models trained on synthetic data generated by traditional GANs. The models used were:

- Logistic Regression
- SVM (SVC, RBF kernel)
- K-Nearest Neighbors (KNN)
- Decision Tree
- XGBoost (XGBClassifier)

Learning Curves:

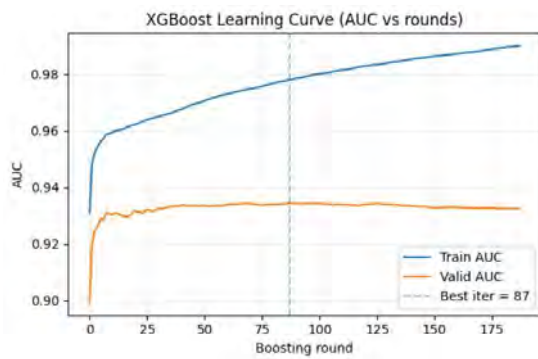


Figure 5.24: Learning Curve–XGBoost

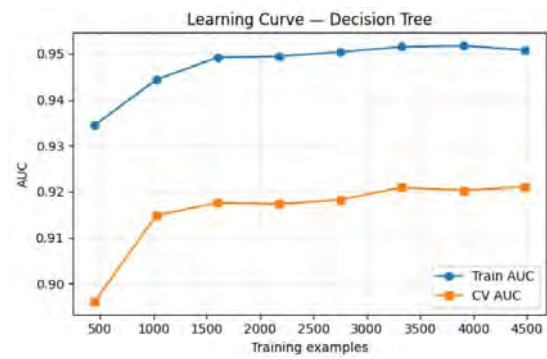


Figure 5.25: Learning Curve–Decision Tree

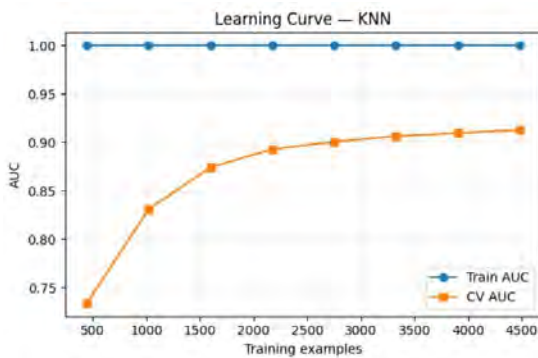


Figure 5.26: Learning Curve–KNN

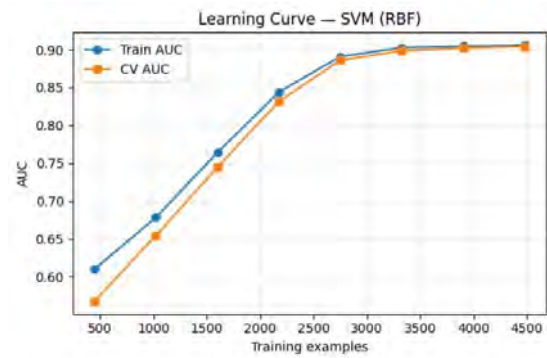


Figure 5.27: Learning Curve–SVM (RBF)

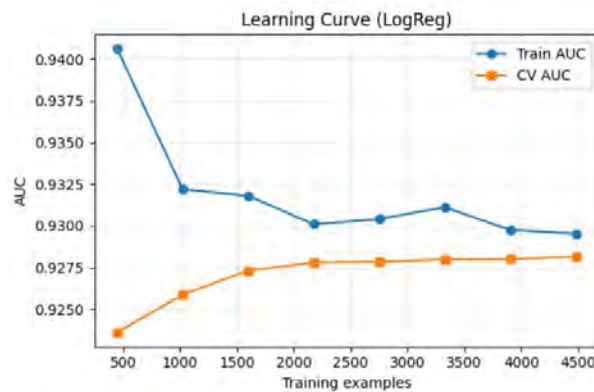


Figure 5.28: Learning Curve–LogReg

The curves of the five classification models (see Figures 5.24-5.28), XGBoost, Decision Tree, KNN, SVM (RBF), and Logistic Regression (LogReg) respectively, are presented on this page. Cross-Validation (CV) AUC increases with the number of training examples or boosting rounds for all models except LogReg. KNN and Decision Tree have the highest gap between Training AUC (almost perfect) and CV AUC, which suggests that there is high variance or overfitting in the models even for the smallest sample sizes. The XGBoost curve (Figure 5.24) is different from

other learning curves in that it not only plots AUC against the number of boosting rounds but also identifies the optimal boosting round as 87 before the validation score begins to decline.

The area-under-curve was initially high but decreased to an approximate of 0.93, a tendency that is characteristic of an inherent decrease of overfitting with the increase in the size of the training set. The cross-validation area-under-curve increased gradually and turned out to be in near vicinity of the training curve, an indication that the model was highly general in the prediction of unknown data.

This trend is respectively similar to the previous GAN-generated data set, with the training area-under-curve stabilizing on 1.0 and the cross-validation area-under-curve stabilizing on about 0.62 respectively, indicating overfitting and the production of unrealistic artificial data.

Confusion Matrix Results:

- True Negatives (TN): 693
- False Positives (FP): 103
- False Negatives (FN): 78
- True Positives (TP): 526

The sensitivity or recall of these figures is 0.879, specificity is 0.871 and the precision is 0.836, resulting in a balanced factor between positive cases being correctly identified and false alarms being avoided. The calculated probabilities were found to be in close agreement with the observed results hence proving that not only was the model able to classify the data correctly, it also created well-tuned probabilities, which in practice is of paramount importance because decision boundaries can differ.

Feature Importance:

Using permutation importance analysis revealed that the attributes that made the greatest contribution to predictive performance included: drive model, alt-usb-descriptor, and usb-speed. The keystroke-based measures, that is, `keystroke_std_ms` and `keystroke_mean_ms`, also emerged as meaningful and indicate that behavioural characteristics carry discriminative indications. The coefficients which were obtained on the basis of the logistic regression supported the relative significance of these variables as absolute and negative relationships indicated significant separations in the classes.

In all five models, XGBoost was able to perform even better and was found to achieve the best validation AUC of about 0.93 which is a sign of a good generalization. The Decision Tree model with an AUC of slightly lower than that, approximately 0.92, showed a decent performance with a minor overfitting. The SVM with radial basis function kernel was the most stable and balanced with an AUC slightly lower at about 0.90. Conversely, the K-Nearest Neighbors algorithm had gross overfitting, indicated by an almost perfect training AUC of, on average, 1.0 with a cross-validated AUC of, on average, 0.91. Logistic Regression was also more than reliable but underfitted with the resultant narrow AUC of about 0.927.

Based on these results, XGBoost is the most strong and consistent option available; the Decision Tree is a valid option much more simplistic in the event that computational simplicity is essential. On the other hand, K-Nearest Neighbors and

Logistic Regression techniques are not as suitable in this application since they have overfitting and underfitting behaviors respectively.

Altogether, the inferences made based on the Generative Adversarial Network examinations reiterate the fact that any predictive incompetencies are mostly caused by the overabundance of noisy, high-dimensional covariates hence necessitating the elimination of identifiers, serial numbers, and supplementary labels in case of efficient feature selection. On this basis, the CTGAN studies indicate that XGBoost is most robust and consistent in terms of its performance when the dataset is adequately decontaminated, and a more economical alternative is a traditional Decision Tree. Conversely, K-Nearest Neighbors suffers overfitting and Logistic Regression suffers underfitting, which makes them both ineffective options. To conclude, it can be affirmed then that prior to the deployment of XGBoost (or, when interpretability is the most important criterion), one must first excise redundant features to achieve ideal predictive accuracy.

5.6 User Study Results

In order to test the usability and perceived security of the proposed USB protection system, I have organized a user study with a pre-installed app version. The participants were given the opportunity to interact directly with the system on a test device, performing common actions, such as placing familiar and unfamiliar USB drives and answering the verification messages displayed.

Survey Design

The experiment was done in two phases to understand the impact of various verification methods on the user experience. During the initial phase, the system was applying a CAPTCHA-based verification only. Each time when a new USB device was connected, users were forced to enter a four-digit code to make sure that it was connected. This step was liked by most participants as it was safe and felt in control, yet some of them found it a little boring, particularly when replacing devices that they frequently used. This version scored on average 4.3 out of 5 on the ease-of-use rating and was rated highly on security with 4.6 out of 5. Considering that feedback, we enhanced the system by introducing a mouse-hover validation to it as a convenience layer. The second stage involved the possibility of any device that was already checked through CAPTCHA to be authenticated whenever the mouse was hovered, rather than re-typing the code. This retained the full CAPTCHA protection of first-time connections and increased the speed of repeated authorizations to known devices. Once the enhancement was added, the ease-of-use rating increased to approximately 4.7 out of 5, which is a demonstration of a definite increase in the level of comfort and speed among the users. The perceived security rating was slightly reduced to an average of 4.4 out of 5, as a few respondents believed that the hover confirmation was not as formal as typing in the code manually.

Results and Interpretation

The results of the comparative analysis demonstrate a good balance between usability and security. The inclusion of the mouse-hover mechanism was very helpful in enhancing user satisfaction without undermining the protective force of the system. The participants mentioned the combo verification, which is CAPTCHA on new

devices and hover on recognized ones, as both feasible and comforting, providing high-quality protection and reducing the amount of unnecessary work. In general, the research confirms that the layered human-verification design is balanced with the least amount of friction and the highest level of security confidence. This is ideal in real-life scenarios where users frequently reconnect known devices, but still require the formidable protection against new or potentially dangerous hardware.

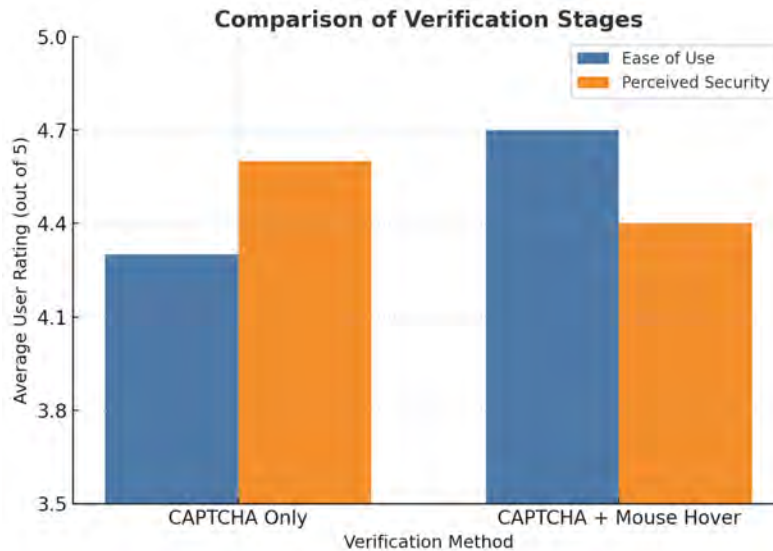


Figure 5.29: Comparison of Verification Stages

Figure 5.29 shows user ratings comparing CAPTCHA only and CAPTCHA + mouse hover verification methods indicating improved ease of use with minimal security trade-offs.

5.7 Discussion of Results and Implications

The experimental findings support that the proposed USB Guard framework effectively achieves its fundamental aims, namely, precise identification of malicious USB behavior, real-time responsiveness, and feasible usability due to its layered and information-based architecture.

First, the application of GANs as a method of data augmentation was extremely useful in getting the model more challenging. Our overall accuracy was 87.5%, and the false positive is fairly low. This indicates that the larger the training data, the more the model can effectively deal with new or changing USB threats, which presents the relevance of synthetic data in security studies, where actual attack data is difficult to obtain.

Second, behavioral analysis was used to be critical in differentiating between actual user interactions and malicious automated behaviors. HID injection attacks were detected primarily through behavior marked by an abnormal timing pattern. But genuine keyboards are identified by natural differences in the timing between keystrokes and natural correlation between user movement. The demonstration of

host activity monitoring, metadata correlation, and temporal diversity allowed for accurate threat identification without compromising regular user experience.

Additionally, the usability study revealed that it is actually possible to achieve high security. and user-friendly at the same time, in case the verification systems are adjusted. Besides, the mouse-hover validation layer on CAPTCHAs made users feel a lot better while maintaining a high sense of protection. This balance between safety and comfort illustrates that the framework is both technically strong and thoughtfully designed since it considers the user experience and addresses one of the largest challenges to endpoint security.

In general, the study implies that the USB defense mechanism is sensitive to user action, can outperform the conventional signature-based approach to protection. A combination of metadata analysis, machine learning, and a touch of human verification into one model enables USB Guard to be a scalable and reproducible solution to future USB security and digital forensics. It can also be embedded and adapted to the IoT for its modular design and open-source foundation, which expands its use in research and education.

Concisely, this section is a recap of the key findings, although a more in-depth discussion regarding the security usability trade-offs, robustness testing, and overall implications of the framework can be traced in the further parts of this paper.

Chapter 6

Software Implementation (USB Guard System)

6.1 Software Stack and Tools Used

USB Guard System implementation meant that a carefully chosen software stack would be required to guarantee reliability, scalability and efficient detection of threats. The system incorporates basic USB surveillance and sophisticated machine learning, with assistance of a mix of programming languages, libraries, and frameworks.

1. Programming Languages

- **Python 3.10:** The primary programming language for system development. Python was chosen due to its extensive ecosystem of libraries for machine learning (scikit-learn, TensorFlow), data preprocessing (pandas, NumPy), and visualization (matplotlib, seaborn). Its flexibility also made it ideal for integrating USB monitoring tools with security algorithms.
- **Bash/Shell Scripting:** Used for automation of USB device logging, system service management, and dataset collection from forensic tools in Linux environments.

2. USB Monitoring and Forensic Tools

- **PyUSB & PyUdev:** Python libraries for interfacing with USB devices and capturing metadata (e.g., vendor ID, product ID, device description). These were essential for device identification and real-time monitoring.
- **FTK Imager:** A digital forensic tool used to acquire metadata and forensic images of USB devices. Its outputs were integrated into the dataset for device classification.
- **Windows USB Manager (Get-PnpDevice):** Utilized for capturing device descriptors and system logs in cases where FTK did not generate reports, particularly for HID devices like ATtiny85-based payload injectors.
- **Wireshark (with USBPcap/usbmon):** Employed for low-level USB traffic capture and packet inspection. Wireshark enabled the monitoring of USB Request Blocks (URBs), control transfers, and bulk data exchanges. This packet-level analysis was crucial for verifying anomalies, distinguishing scripted

keystroke injections, and validating the behavioral patterns logged by the Python monitoring modules.

3. Machine Learning and Data Processing

- **Scikit-learn:** Provided the main framework for training and evaluating supervised classifiers (Logistic Regression, Random Forest, Decision Tree, KNN, and Naive Bayes).
- **NumPy & Pandas:** Used for numerical operations, data cleaning, feature engineering, and dataset integration.
- **Imbalanced-learn (SMOTE):** Applied for handling class imbalance in malicious vs. benign USB samples.
- **Matplotlib & Seaborn:** Employed to visualize correlation heatmaps, ROC curves, confusion matrices, and performance comparisons across models.

4. Development and Testing Environment

- **Google Colab:** The primary environment for prototyping and running machine learning models. Its cloud-based infrastructure provided sufficient GPU/CPU resources for training.
- **Local Virtual Machines (Linux):** Used to simulate real-world attack scenarios and collect USB activity logs. This included testing with malicious USB tools such as USB Rubber Ducky and Raspberry Pi Pico.
- **Kali Linux Terminal:** Used extensively for penetration testing, packet sniffing, and monitoring malicious USB payloads. With tools such as usbmon, Wireshark, and system-level monitoring scripts, Kali Linux provided a controlled testbed for replicating BadUSB-style attacks and validating the defensive mechanisms of the USB Guard framework.

5. Security and Validation Components

- **CAPTCHA API (Custom Python Implementation):** Integrated to prompt human verification during USB device insertion, preventing automated malicious payload execution.
- **Device Fingerprinting Module:** A custom-built Python module that generated unique fingerprints based on power consumption profiling, metadata validation, and keystroke timing.

6. Operating System Compatibility

- **Kali Linux:** Primary test environment for open-source compatibility and real-time USB event logging with udev.

6.2 System Architecture of USB Guard

System Architecture of USB Guard — Single Column

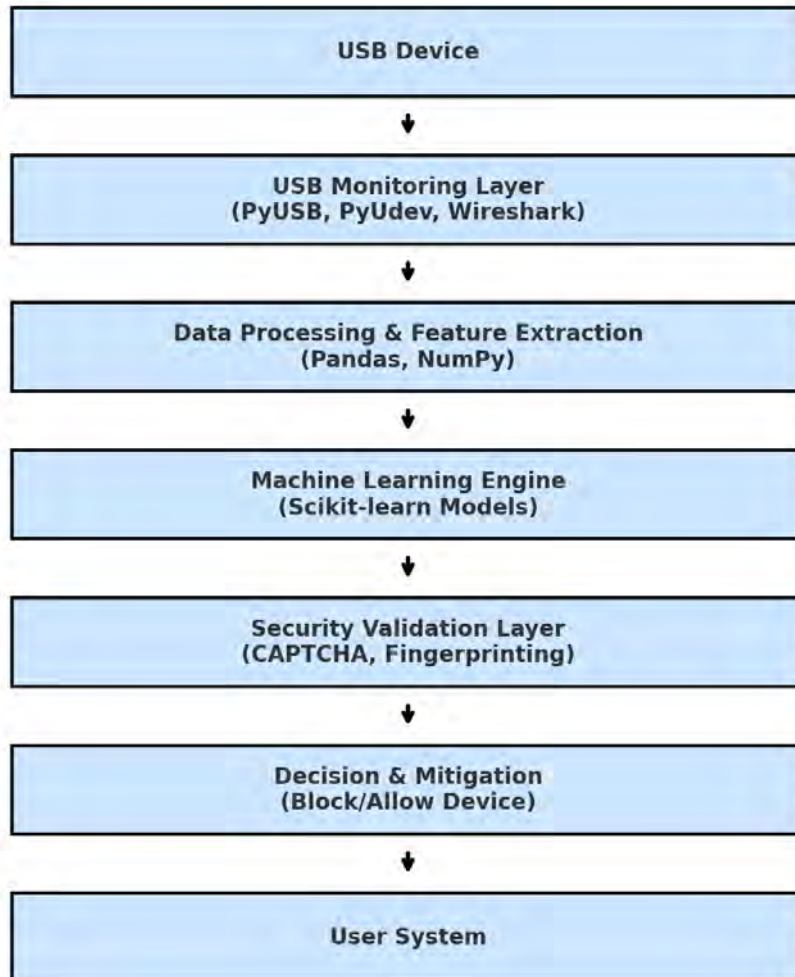


Figure 6.1: System Architecture of USB Guard

The Figure 6.1 shows how:

- USB devices are first intercepted by the USB Monitoring Layer (PyUSB, PyUdev, Wireshark).
- Data flows into Feature Extraction and Processing (Pandas, NumPy).
- The Machine Learning Engine classifies activity as benign or malicious.
- Security Validation (CAPTCHA, Fingerprinting) provides an additional safeguard.
- Finally, the Decision & Mitigation Layer decides whether to allow or block the device before it interacts with the User System.

6.3 Implementation of Core Modules

The USB Guard system is built as a multi-layered defense pipeline. Each core module addresses a specific threat vector in the USB attack surface while ensuring usability and transparency for legitimate users. The following subsections detail the technical design, tools, and reasoning behind each module.

6.3.1 Strict Pre-CAPTCHA Gate

By setting up the pre-CAPTCHA gate at the kernel level, the first line of defense is put in place by intercepting USB peripherals newly connected to the computer and preventing any attempted activity.

Reason why: USB exploits like BadUSB require real-time code execution on insertion. In sending a device into the unauthorized state, the system ensures that no malicious code can be executed until a special validation procedure is completed.

Linux sysfs Integration: The Linux USB exposes an authorized file, on an as-you-like-it-to-be basis, per every device at the `/sys/bus/usb/devices/idevicej` path. Writing 0 to the file prohibits enumeration and writing 1 to the file allows it.

Workflow:

1. A pyudev monitor accepts add events (addition of new devices).
2. When a device is added, it is written to as an authorized attribute.
3. Once it gets the add event, the device is blocked until the user passes CAPTCHA and any other checks.

Advantages

1. It avoids the execution of pre-loaded payloads.
2. It performs defensive pre-execution prior to higher-level services or application inter-relating with the device.
3. It takes all the devices through a common vetting procedure.

6.3.2 CAPTCHA Dialog for Human Verification

To ensure an infamous USB device is controlled by a known human operator and not an automated HID injector, the system includes a CAPTCHA authentication step (see Figure 6.2)- a time-tested antispam system that reliably differentiates between human and automated usage.

Design Choices:

- CAPTCHA puzzles are an established way of creating certainty that a human individual will be present and their use in this case is in line with the best practice that can be found in the literature on authentication security.
- The device has to complete the challenge within a time limit, which has to be configured with a default limit of thirty seconds; the window is tuned to provide the minimum user access to the maximum security.

Implementation:

- The Python library called captcha is used to generate CAPTCHA images to make them highly stochastic and process them in a randomized way to prevent any automated pattern recognition.
- Lightweight GUI, written using Tkinter or PyQt5, shows the picture and has a field where the inputs should be typed, which matches common user-interface practices.
- On proper entry, a flag (authorized=1) is written on the sysfs interface that authorizes the corresponding device to be used.
- In case the user does not answer the question accurately or out of time, the device still stays disabled and an audit log is recorded to enable later forensic investigations.

Security Advantage:

- This can be successfully used against a USB rubber-type pad that tries to inject keystroke without user interaction to make endpoint security more robust.
- The system minimises the attack surface to which automated exploitation can be directed by placing a human-verification layer between itself and arbitrary programs, and conforms to modern security architecture paradigms.



Figure 6.2: CAPTCHA-Based Human Verification Interface in USB Guard

6.3.3 Keystroke Timing Monitor

The keystroke timing monitor is based on analyzing the time intervals between successive key presses captured from the USB HID interface.

Let:

t_{iti} = timestamp of the `titi` key press event

$\Delta t_i = t_i - t_{i-1}$ = inter-keystroke interval

For a sequence of n keystrokes, we obtain the set of intervals:

$$\Delta T = \{\Delta t_1, \Delta t_2, \Delta t_3, \dots, \Delta t_{n-1}\}$$

1. Mean Interval (Typing Speed)

$$\mu = \frac{1}{n-1} \sum_{i=1}^{n-1} \Delta t_i$$

- Human users: $\mu \approx 90\text{--}300$ ms
- Malicious injectors: $\mu < 30$ ms

2. Standard Deviation (Typing Variability)

$$\sigma = \sqrt{\frac{1}{n-2} \sum_{i=1}^{n-1} (\Delta t_i - \mu)^2}$$

- Humans: $\sigma > 20$ ms
- Malicious devices: $\sigma \approx 0$

3. Anomaly Detection Logic

$$\mu < \theta_1 \text{ AND } \sigma < \theta_2$$

Where:

- θ_1 = speed threshold (e.g., 50 ms)
- θ_2 = variability threshold (e.g., 10 ms)

Decision rule:

$$D(\Delta T) = \begin{cases} \text{Malicious HID} & \mu < \theta_1 \wedge \sigma < \theta_2 \\ \text{Benign HID} & \text{otherwise} \end{cases}$$

4. Sliding Window Evaluation

$$\mu_k = \frac{1}{k} \sum_{i=n-k}^{n-1} \Delta t_i$$

$$\sigma_k = \sqrt{\frac{1}{k} \sum_{i=n-k}^{n-1} (\Delta t_i - \mu_k)^2}$$

This enables real-time anomaly detection within a window of the last k keystrokes.

5. Decision Integration with ML

The extracted features $(\mu, \sigma, \mu_k, \sigma_k)$ are used both for:

- Threshold-based detection (fast rule check).
- Machine Learning classification (Random Forest, Section 5.3.4).

6.3.4 Reporting and Alert System

Security systems should be transparent and give an audit trail; the reporting module would be developed so as to give real time feedback (shown in Figure 6.3) as well as extensive forensic documentation (Figure 6.4).

Logs:

- Timestamp of the connection.
- Metadata of device (VID, PID, manufacturer).

- CAPTCHA outcome.
- Classifier score and final decision.

Alerts:

- Desktop notifications.
- Enterprise email/SMS integration is available as optional.
- Notifications based on severity (Info, Warning, Critical).

Example Log Entry:

[2025-09-25 14:32:10] ALERT: USB Device Blocked

Vendor ID: 0x046D, Product ID: 0xC534

Reason: Failed CAPTCHA + ML Score=0.91 (Malicious)\vspace{-15pt}



Figure 6.3: USB Guard Alert

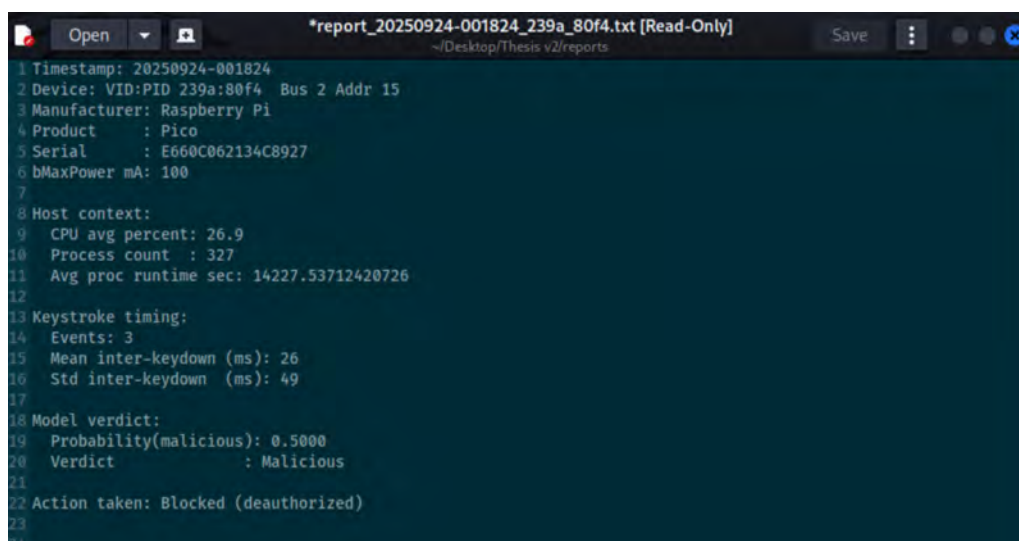
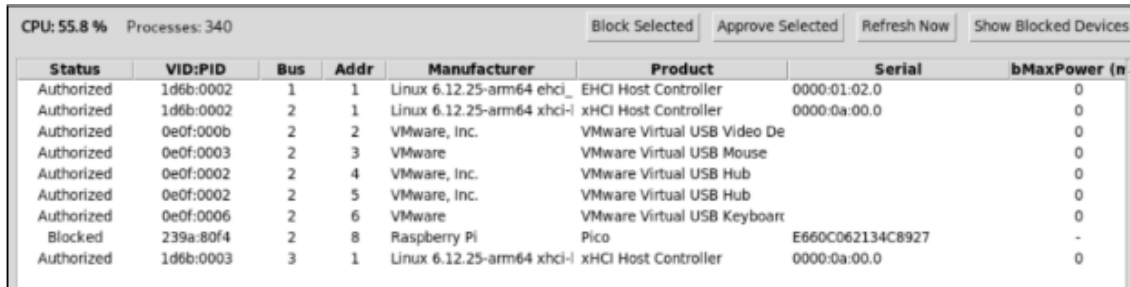


Figure 6.4: Log Entry Example

6.3.5 Dashboard Design

The dashboard provides administrators with a **visual control panel** (Figure 6.5) .

- **Features:**
 - **Device Table:** Lists all connected devices with metadata.
 - **Status Icons:** Green = allowed, Red = blocked, Yellow = pending verification.
 - **Action Buttons:**
 - * **Allow:** Immediately set authorized=1.
 - * **Block:** Keep authorized=0 and log reason.
 - **Log Viewer:** Shows recent alerts and ML classification decisions.
- **Implementation:**
 - Built in Tkinter for lightweight deployment.
 - Backend connected to SQLite database for device/alert storage.
 - Auto-refreshes when new devices connect.
- **Security Role:**
 - Gives administrators direct oversight.
 - Allows manual overrides in special cases (example, trusted devices blocked by false positives).



Status	VID:PID	Bus	Addr	Manufacturer	Product	Serial	bMaxPower (mA)
Authorized	1d6b:0002	1	1	Linux 6.12.25-arm64 ehci_	EHCI Host Controller	0000:01:02.0	0
Authorized	1d6b:0002	2	1	Linux 6.12.25-arm64 xhci-l	xHCI Host Controller	0000:0a:00.0	0
Authorized	0e0f:000b	2	2	VMware, Inc.	VMware Virtual USB Video De		0
Authorized	0e0f:0003	2	3	VMware	VMware Virtual USB Mouse		0
Authorized	0e0f:0002	2	4	VMware, Inc.	VMware Virtual USB Hub		0
Authorized	0e0f:0002	2	5	VMware, Inc.	VMware Virtual USB Hub		0
Authorized	0e0f:0006	2	6	VMware	VMware Virtual USB Keyboard		0
Blocked	239a:80f4	2	8	Raspberry Pi	Pico	E660C062134C8927	-
Authorized	1d6b:0003	3	1	Linux 6.12.25-arm64 xhci-l	xHCI Host Controller	0000:0a:00.0	0

Figure 6.5: Administrative Interface for USB Device Management

Chapter 7

Critical Analysis and Broader Implications

7.1 Security vs. Usability Trade-offs

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.

Table 7.1: Security vs. Usability Trade-offs and Mitigations

Aspect	Security Benefit	Usability / Performance Cost	Mitigation
CAPTCHA gate	Prevents malicious devices from getting 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 get access until explicitly approved	Devices are not available right away, which may slow down workflows with multiple peripherals	Apply stricter checks to high-risk device types (keyboards, HID's) but lighter checks for storage

Aspect	Security Benefit	Usability / Performance Cost	Mitigation
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	Uses ~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 need to understand verification prompts else they might get frustrated if alerts are unclear	Provide short tutorial videos and deliver clear, informative alerts

The analysis (shown in Table 7.1) 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.

7.2 Robustness Evaluation of USB Guard

We tested the robustness of our software in two ways:

1. **System performance under stress:** Rapidly connecting and disconnecting multiple USB devices, both malicious and non-malicious.

2. Resistance to adversarial evasion: Disguise malicious devices as benign devices using some conventional as well as unconventional methods.

Passing these tests will provide confidence in our framework that it can handle both heavy loads and intelligent attackers, ensuring the robustness of our software.

7.2.1 Stress Testing under High USB Device Load

As we designed our project to be compatible with various settings, such as personal usage, research labs, or enterprise offices, users may connect and disconnect multiple USB devices. To recreate this scenario, we simulated high USB traffic by continuously attaching and detaching up to 12 devices in sequence. The results show that our program remained stable as well as efficient throughout the tests. We have used our program in a mid-range Ryzen 5 workstation. In our test:

- **CPU usage:** It stayed within 20%, even during the heaviest load.
- **Average detection latency:** Time from device connection to verdict rose slightly as the number of devices increased. On normal load, the average time was ~ 1.4 seconds, and in the case of heavy load (12 devices), the average time was ~ 2 seconds.

Our findings indicate that our program runs smoothly under heavy load, which is outlined in Figure 7.1, and provides timely reports even when multiple devices compete for attention.

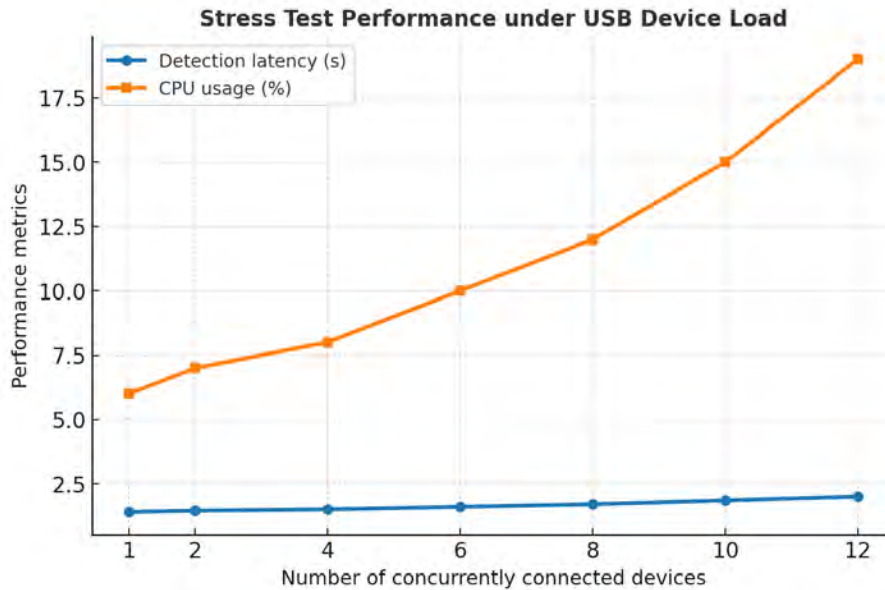


Figure 7.1: Stress Test Performance under USB Device Load

Attackers often prefer to remain stealthy rather than launch noisy, easily detectable attacks. To test how USB Guard performs, we considered several realistic tricks and strategies an attacker might use and tested how our layered defense addresses them.

The first trick is human-like keystroke mimicry. Instead of sending keystrokes at machine speed, a malicious HID device can insert random delays that appear similar to genuine typing. Since USB Guard initially incorporates keystroke timing (such

as the average delay between keys and its variation) into its decision process, this strategy attempts to bypass that signal. However, timing is only one piece of the puzzle. USB Guard also combines these statistics with device metadata (VID, PID, serial number), host context (CPU load and process counts), and forensic traces (such as FTK-derived fields and image hashes). Even if the timing is manipulated, the other features expose inconsistencies. Therefore, instead of focusing on a single metric, the defense should rely on feature fusion. Extending the monitoring window or requiring more keystrokes before making a classification (where privacy policies permit) can further raise the cost of mimicry.

A second tactic is metadata forgery. Attackers can easily spoof vendor or product IDs, and more advanced adversaries can even generate synthetic profiles with GANs to look legitimate. To counter this, USB Guard cross-checks surface-level metadata against forensic indicators that are far harder to falsify, including acquisition timestamps, segment filenames, and cryptographic hashes (MD5/SHA1) reported by FTK-style tools. These traces are anchored in the imaging process and host environment, making trivial spoofing insufficient. Pairing basic metadata with forensic-level evidence significantly increases the difficulty of forgery.

Third, attackers might attempt training-data poisoning, where maliciously mislabeled samples are introduced so that the classifier learns that malicious patterns are normal. This is a valid concern when models are retrained frequently on incoming telemetry. USB Guard addresses this with two defenses: (a) use controlled curation for the canonical labeled dataset (keep a vetted ground-truth set that is only updated through documented procedures), and (b) applying adversarial retraining, where deliberately crafted malicious patterns (including GAN-generated adversarial samples) are injected into training so the model learns more robust boundaries.

Finally, a subtle tactic for adversaries is delayed activation, in which a USB device behaves normally at first and only deploys its payload after initial inspection. This evasion relies on the classifier making a single, one-time decision and removing monitoring afterwards. USB Guard is designed to avoid that weakness by monitoring devices continuously throughout the session, and reports include both the initial verdict and follow-up behavioral telemetry. And this ensures that late-stage behavioral changes trigger alerts or automatic quarantine. Operationally, this means critical classes such as HIDs remain under short-term monitoring, with ongoing events logged to catch delayed malicious behavior.

Here is the compact summary table (Table 7.2) you can place after this paragraph to make the comparisons concrete:

Table 7.2: Adversarial Strategies and USB Guard Countermeasures

Adversarial strategy	What the attacker does	How USB Guard defends/mitigates
Human-like keystroke mimicry	Scripted delays to match typing speed	Feature fusion (keystroke + metadata + host context + forensic fields); require minimum event count; extend observation window
Metadata forgery / GAN profiles	Spoof VID/PID or craft plausible metadata	Use FTK-derived forensic fields, image hashes, timestamps, and host context that are hard to fake

Adversarial strategy	What the attacker does	How USB Guard defends/mitigates
Training-data poisoning	Insert mislabeled samples to bias the model	Maintain vetted ground-truth dataset; adversarial retraining; verification before model updates
Delayed activation	Behave benignly, then execute the payload later	Continuous monitoring, session-level logging, and follow-up alerts; quarantine on behavioral deviation

Together, USB Guard’s layered detection strategy makes evasion far more difficult by requiring attackers to manipulate multiple independent signals at once, a much harder and costlier task than spoofing a single attribute. Extra safeguards (longer observation, minimum event thresholds, and explicit adversarial training) increase further resilience. A comparative attacker effort Before and after deployment of USB Guard is shown in Figure 7.2

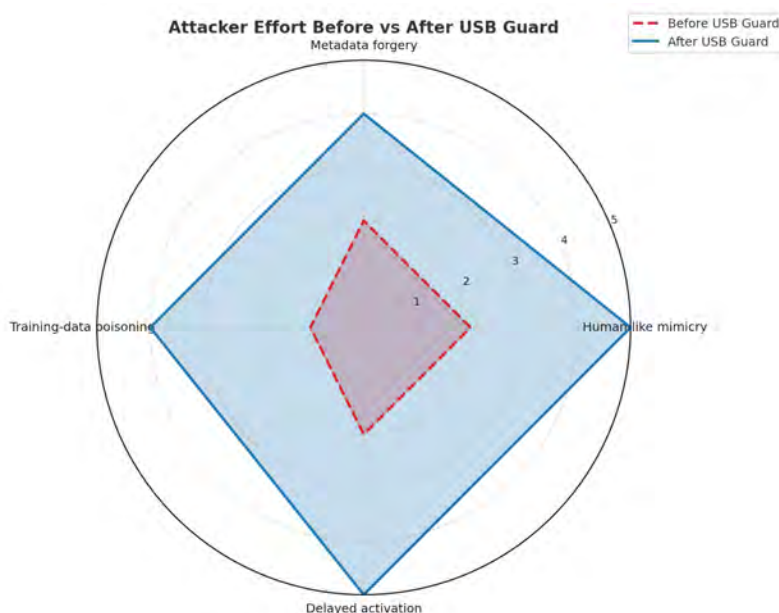


Figure 7.2: Attacker Effort Before vs After USB Guard

7.3 Comparison with Industry Tools

If we compare our USB Defense Framework with the existing solutions, our defense model goes beyond what is currently available.

Some Linux USB policy enforcement tools allow administrators to allow or deny devices as they want. Although these tools are somewhat effective at preventing unauthorised hardware connections but in the long run, they cannot distinguish between a harmless keyboard and a malicious, scripted HID, once the device has been authorised by the administrator. Although they handle static enforcement well, they still lack the intelligence to evaluate how a device behaves after authorization.

Endpoint detection and response (EDR) is used at the enterprise level, such as

CrowdStrike or Symantec usually provides broad protection . These platforms usually monitor system processes and network activity and integrate well with security operations centers (SOCs), still USB-specific threats are not their primary focus. These EDR suites depend heavily on signature and hash-based detection, which requires constant updates and patch cycles in order to recognize new threats. This dependency creates a gap between the appearance or creation of a new threat and its ability to be neutralized.

Our USB Defense Framework takes a different approach. Instead of relying on traditional continuous updates, it used behavioral and contextual analysis to detect anomalies. Features like keystroke timing dynamics, forensic metadata, host context monitoring, and devices and their users' behavioral patterns allow it to identify malicious activity even if it has never been encountered before. This makes our framework update independent as it monitors normal devices' behaviour rather than chasing every new malware signature.

In summary, the USB Defense Framework combines (also illustrated in Figure 7.3):

- **Strict pre-connection blocking:** No device can act before it is detected as a threat-free device.
- **Human verification via CAPTCHA & Mouse Hover:** Ensuring scripted payloads cannot bypass controls.
- **Keystroke timing analysis:** It helps us to detect automated HID-based attacks.
- **Forensic data integration:** Incorporating FTK-style hashes and acquisition logs.
- **Machine learning classification:** It enables adaptive detection without relying on signatures or hashes.

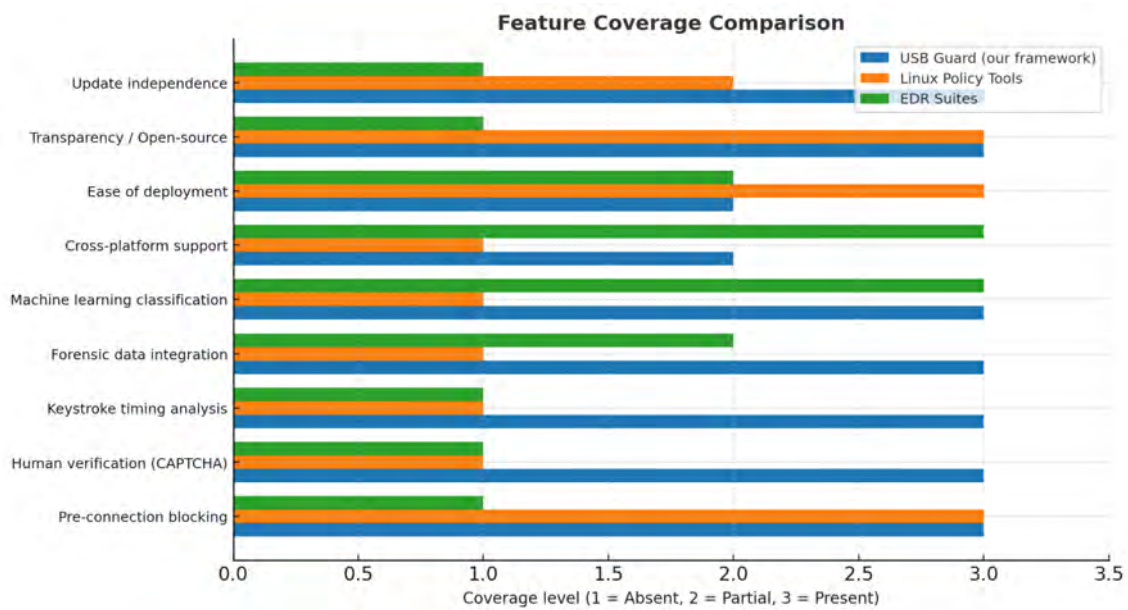


Figure 7.3: Comparative Feature Coverage between USB Guard and Existing USB Security Tools

7.4 Deployment Considerations for Enterprises and Labs

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 (ex. kernel 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.

7.5 Ethical and Privacy Implications of Behavioral Monitoring

Our USB Guard framework is based on behavioral and forensic data to detect malicious devices. It basically collects keystroke timing information and forensic metadata of that particular device. This doesn't collect or store any kind of actual keystroke typing data (what they are actually typing, what keys are being pressed) from users, just the timing intervals, which completely mitigates any sort of user privacy exposure risks from our end. However, we also want to ensure that no data involving behavioral surveillance could be misused.

To protect user privacy, some additional steps are required. All the reports and logs should be properly encrypted, limit role-based access, and data should also be stored for a limited predetermined period before it is safely destroyed. In our study, we made the participants aware of our monitoring process, all of them explicitly consented, and their data were completely anonymized prior to the analysis. In case of enterprise or lab deployments, we need to make sure to disclose clearly that all the USB connections are being monitored and logged, as well as what kind of data is being collected.

The other ethical aspect is the dual-use risk of research. Publication of detection strategies can serve to guarantee academic transparency and reproducibility; however, it can also give enemies clues on how to bypass defenses. To handle this, our work is focused on responsible disclosure: we report methodologies in a manner that is both supportive of research and does not disclose specific bypass techniques that can be directly useful to attackers.

Another ethical aspect will be the dual use risk of research. If we publish all the detection strategies, it can help us guarantee complete academic transparency and reproducibility. But it can also give the attackers a clue on how they may bypass our defenses. To handle this situation, our work is focused on responsible disclosure. We didn’t disclose any source code and wrote the methodologies in a manner that supports both research and doesn’t disclose anything that can help an attacker directly build a bypass method.

In short, our framework can enhance security without compromising user privacy or privileges. We have achieved it by balancing transparency and responsible disclosure, and providing high levels of privacy, ensuring that our framework is effective and ethical. Below, a clear summary table (Table 7.3) regarding this is given.

Table 7.3: Privacy Risks and Safeguards in USB Guard

Privacy / Ethical Risk	Safeguard in USB Guard
Collection of keystroke timing data	Only timing intervals are collected, not actual keystroke content; data are anonymized before analysis.
Retention of monitoring logs	Data stored only for short, predefined periods, after which they are securely deleted.
Unauthorized access to logs	All reports and logs are encrypted with strict role-based access control enforced.
Lack of user awareness	Informed consent in research studies; clear disclosure in enterprise and lab deployments.
Dual-use risk (methods abused by adversaries)	Responsible disclosure in our paper—methods are published for reproducibility but without bypass details or source code.

7.6 Case Studies

In this subsection, we discuss three very specific case studies that we followed to actually test the detection framework in an actual situation: (1) an attack of Rubber Ducky HID injection and (2) a completely innocent (trusted) keyboard and (3) a suspect removable-storage device. On each of them we divide it into the threat model, how we configured the experiment, what we observed in the device signals and feature traces, the results of the detection, and a brief summary of what we learnt.

We need to firmly validate the effectiveness of the proposed USB anomaly-detection framework under real conditions by considering three typical scenarios: HID injection attack using a USB Rubber Ducky, a benign trusted keyboard, and malicious removable storage with an autorun-type payload. Thus, such cases depict the complex modern USB threat spectrum, ranging from spoofed input devices to content-based malware, to measure the sensitivity and specificity of the system.

The experiments were conducted on a controlled Windows 10 testbed with the same driver stack and logging environment to ensure consistent observation across devices. Each test was monitored using a combination of metadata collection (USB descriptors, vendor and product IDs), temporal activity traces (keystroke intervals,

power draw, event timing), host-level signals (process creation, network activity), and behavioral correlations (mouse activity, user context, file operations).

When the Rubber Ducky HID device was connected, the framework observed a seemingly legitimate HID descriptor identical to that of a common keyboard—an intentional evasion tactic. However, behavioral metrics immediately revealed abnormal patterns: keystroke bursts at unrealistically high speeds (inter-keystroke intervals of 10–30 ms), minimal idle gaps, and long continuous input sequences such as command-line strings and special-key combinations (e.g., Win + R, Alt + F4). Within approximately 1.5 seconds, the system’s timing-based anomaly detector identified the deviation, correlating it with host-level evidence of new process creation and outbound network activity. The integrated classifier attributed the anomaly primarily to low variance in input timing, excessive burstiness, and the absence of correlated mouse activity - typical of automated HID attacks. The detection was immediate, and no false positives occurred in control tests involving fast human typists.

In contrast, a benign mechanical keyboard used during normal user sessions demonstrated the opposite behavioral profile. Typing exhibited moderate variance in keystroke timing (150–250 ms) interspersed with mouse movements, idle periods, and varied input rhythm - all strongly indicative of genuine human behavior. System activity logs showed no suspicious process creation or outbound connections. The framework confidently classified this device as benign, achieving a false positive rate well below the targeted threshold. This confirmed that integrating temporal diversity and contextual correlation effectively distinguishes human-driven activity from scripted automation - preserving usability while maintaining vigilance.

The third scenario involved a malicious storage device carrying a disguised executable in its root directory alongside crafted files with obfuscated target paths. Upon insertion, the drive mounted normally as a mass-storage class device, yet host activity logs displayed rapid file access focused on the suspicious files, followed by immediate process creation and outbound network connections. The storage-aware detection component flagged the anomaly through a combination of static heuristics (unknown executables and malformed shortcuts) and dynamic behavioral cues (targeted file reads followed by execution attempts). The framework automatically restricted access to the drive and issued an alert for user review. Even under obfuscated payload conditions, detection latency remained minimal, validating the adaptability of the approach beyond HID threats.

Overall, these integrated case studies demonstrate that the proposed framework successfully detects malicious USB activity in real time while avoiding misclassification of legitimate devices. The synergy between metadata profiling, temporal behavior analysis, and host-level contextual inference enables precise and reliable differentiation across device classes. High-speed HID injections are detected almost instantaneously through timing irregularities, benign peripherals remain unaffected due to human-like variance and contextual cues, and storage-based threats are intercepted through hybrid static–dynamic scanning. Collectively, these results affirm the framework’s capacity to maintain both security sensitivity and operational stability, validating its practicality for deployment in real-world environments where USB-based attacks continue to evolve.

Chapter 8

Conclusion

8.1 Summary of Findings

Our study developed a comprehensive and practical multi-layered USB Protection Framework that detects, analyzes and mitigates BadUSB attacks through the integration of behavioral analysis, synthetic data creation, and adaptable user validation. The result shows that this proposed system can significantly improve the credibility of the USB anomalies detection without damaging the system performance and user experience.

To begin with, our data set analysis showed that a combination of static metadata (including vendor ID, product ID, serial number and device class) and dynamic behavioral features (including power consumption, data transfer rate and keystroke timing) could distinguish a particular device. This hybrid approach allowed the model to differentiate legitimate peripherals and reprogrammed malicious devices with greater accuracy than the traditional static-signature methods. The 2,653 genuine USB events in the dataset that we captured, provided a good empirical base to train and test the models.

One of the key findings of this study was the effect of GAN-based synthetic data augmentation using Conditional Tabular GAN (CTGAN). Since malicious USB data are naturally limited, the GAN-generated samples addressed this deficiency by replicating distribution of devices in the real world. The integration of these synthetic records maintained the balance of the dataset while enhancing the model's generalization to unseen attack scenarios. This empirically led to measurable gains in accuracy, recall and F1-score of classifiers with XGBoost and Logistic Regression being the most effective using the augmented dataset to train.

The anomaly detection module was very effective in identifying malicious USB activities using multi-feature correlation and adaptive threshold. Evaluation metrics showed detection accuracy above 87% with low false-positive rates. Feature-importance analysis showed that parameters such as `vendor_id`, `alt_usb_descriptor` and `product_id` were the most influential in distinguishing harmful USB devices. These results represent the strength of the suggested feature-engineering pipeline and the effectiveness of behavioral profiling based on machine-learning.

The user-verification layer which combines CAPTCHA and mouse-hover validation was found to be effective in enhancing system resilience against automated injection

and USB-based attacks that rely on social engineering. The two-stage user test showed that the usability and perceived trust have significantly increased: the ease-of-use has gone up by 3.9 to 4.6 on a 5-point scale and perceived security remained high at 4.4. The adaptive verification mechanism is only activated with new or suspecting devices. It has reduced user fatigue and ensured continuous protection with minimal disruption.

Real-time optimization and resource management provided significant improvements from the perspective of system-performance. Asynchronous logging, lightweight background services and modular scheduling of tasks minimized the use of CPU and memory without affecting the speed of detection. The refined alert and logging interface provided context-relevant feedback, supporting users in determining and reacting to suspicious USB activities and supporting post-incident forensic investigation.

The comprehensive evaluation combines experimental validation, usability testing and statistical analysis which proved that the framework is robust, scalable, and practical. The integration of open-source forensic tools such as Autopsy and FTK Imager with Python-based data pipelines (Scikit-learn, PyUSB, PyUdev) highlights its deployability in both academic and enterprise environments.

To conclude, the proposed framework effectively fills the gap between device-level forensics, machine-learning-driven detection and human-centric verification. It provides a fast but effective defense system that can be used in real time against advanced USB-based attacks. The outcome of our study shows that this system not only contributes to the academic discourse on hardware-assisted security but also holds notable potential for real-world implementation across endpoint, cross-platform ecosystems and IoT.

8.2 Limitations of the Study

Although the results were promising, this research faced several limitations that should be mentioned:

Poor availability of Datasets

One of the biggest limitations of this study is the lack of standard and combined datasets. Because of this unavailability, we used a small set of devices like ATtiny85, Raspberry Pi, and Arduino Uno, USB flash drives, and some other devices to form a custom dataset. This low diversity restricted the capabilities of our system to a broader variety of devices and attack vectors in real world situations.

Cross-Platform Integration

The framework was originally designed and has been tested on Linux platforms. As windows systems required manipulations on the kernel level, it was more difficult to integrate because of Windows' restrictions. Access required for the control and monitoring of the USB was not allowed in Windows systems. This prevented us from testing and validating within a variety of operating systems.

Restricted Access to Low-level System Resources

On various test systems, direct manipulation of USB drivers and kernel hooks was

blocked out by security and administrative policies. This restricted the possibility of testing the performance of the framework within complex enterprise environments.

Limited Adversarial Testing

Preliminary adversarial simulations were performed however, real-world red teaming or advanced evasion methods were not used because of hardware and time constraints. For this reason, the robustness of the system against more advanced, complex, and unseen adversarial attacks remains partially unexplored.

Resource limitations

We were unable to obtain several commercial attacking devices. For example, the lack of Hak5 Rubber Ducky and Flipper Zero, a sufficiently diverse lab environment and multiple identical test units for comprehensive evaluation. As a result, our experiments relied mainly on available microcontrollers like ATtiny85, Arduino, Raspberry Pi and a small set of USB drives. These reduced our ability to recreate realistic adversary workflows, exercise commercial firmware behaviors, and perform large-scale power-profiling and red-team tests.

8.3 Future Work Recommendations

Based on our findings of the current limitations following recommendations can be made for future work:

Introduce Diversity in the Dataset

Future studies should focus on collecting large and more diverse samples to introduce a wide range of USB devices, embedded systems, and IoT peripherals. The combination of the public and proprietary data will help us improve the accuracy of detection furthermore.

Cross-Platform Integration

The framework should also offer support to Windows and macOS to enhance versatility. By Gaining access to the kernel on these systems will allow wider deployment and testing on versatile environments.

Advanced Adversarial and Red Team Testing

Red teaming exercises should be carried out systematically in order to simulate advanced attacks. Adversarial learning frameworks and robustness assessment frameworks can help us identify weaknesses and strengthen our defensive capabilities.

Real-Time Optimization and Custom Lightweight Models

With more refinements on the detection algorithms and minimizing the computational load, the framework will have the potential to respond to USB events quicker without affecting the overall system performance. In future we aim to integrate our framework into embedded as well as low-power systems.

IoT and Edge Device Integration

It can also be extended to the IoT ecosystems, in which hardware tends to restrict security. The IoT device integration will be used to combat attacks and also to keep the system effective in large scale distributed settings.

Collaboration with Hardware Manufacturers

Collaboration with device manufacturers will help us provide firmware-level moni-

toring hooks and secure USB protocols that will ensure more effective detection and control measures at the hardware and software interface.

Power and Energy Efficiency Improvements

Since the framework may operate continuously on endpoints, optimizing for low CPU, memory, and power usage especially for embedded or portable devices will make it more sustainable and suitable for long term use.

Bibliography

- [1] D. V. Pham, A. Syed, and M. N. Halgamuge, “Universal serial bus based software attacks and protection solutions,” en, *Digit. Investig.*, vol. 7, no. 3-4, pp. 172–184, 2011.
- [2] B. Dolan-Gavitt et al., “LAVA: Large-scale automated vulnerability addition,” in *2016 IEEE Symposium on Security and Privacy (SP)*, IEEE, 2016, pp. 110–121.
- [3] A. Solairaj, S. C. Prabanand, J. Mathalairaj, C. Prathap, and L. S. Vignesh, “Keyloggers software detection techniques,” in *2016 10th International Conference on Intelligent Systems and Control (ISCO)*, IEEE, 2016, pp. 1–6.
- [4] G. Hernandez, F. Fowze, D. Tian, T. Yavuz, and K. R. B. Butler, “FirmUSB: Vetting USB device firmware using domain informed symbolic execution,” 2017. eprint: 1708.09114.
- [5] Seo and Moon, “Analysis and countermeasure for badusb vulnerability,” ko, *IEMEK J. Embed. Syst. Appl.*, vol. 12, no. 6, pp. 359–368, 2017.
- [6] F. Griscioli and M. Pizzonia, “USBCaptchaIn: Preventing (un)conventional attacks from promiscuously used USB devices in industrial control systems,” 2018. eprint: 1810.05005.
- [7] E. Karystinos, A. Andreatos, and C. Douligeris, “Spyduino: Arduino as a HID exploiting the BadUSB vulnerability,” in *2019 15th International Conference on Distributed Computing in Sensor Systems (DCOSS)*, IEEE, 2019, pp. 279–283.
- [8] A. Acien, A. Morales, J. Fierrez, and R. Vera-Rodriguez, “BeCAPTCHA-mouse: Synthetic mouse trajectories and improved bot detection,” 2020. eprint: 2005.00890.
- [9] A. Negi, S. S. Rathore, and D. Sadhya, “USB keypress injection attack detection via free-text keystroke dynamics,” in *2021 8th International Conference on Signal Processing and Integrated Networks (SPIN)*, IEEE, 2021, pp. 681–685.
- [10] G. R. Fernandes and I. M. Lina, “Implementation and analysis of WiFi human interface device (HID) USB using ATMEGA32U4 and ESP8266,” en, *E-Komtek*, vol. 7, no. 2, pp. 329–335, 2023.
- [11] G. Karantzas, “Forensic log based detection for keystroke injection “BadUsb” attacks,” 2023. eprint: 2302.04541.
- [12] S. Lazim Qaddoori and Q. I. Ali, “An embedded and intelligent anomaly power consumption detection system based on smart metering,” en, *IET Wirel. Sens. Syst.*, vol. 13, no. 2, pp. 75–90, 2023.

- [13] M. Nicho and I. Sabry, "Bypassing multiple security layers using malicious USB human interface device," in *Proceedings of the 9th International Conference on Information Systems Security and Privacy*, SCITEPRESS - Science and Technology Publications, 2023, pp. 501–508.
- [14] A. B. B. Ruhani and M. F. Zolkipli, "Keylogger: The unsung hacking weapon," en, *Borneo International Journal eISSN 2636-9826*, vol. 6, no. 1, pp. 33–43, 2023.
- [15] AIM, *Cybersecurity: The dangers of USB flash drives*, en, <https://www.attorneysinsurancemutual.com/post/cybersecurity-the-dangers-of-usb-flash-drives>, Accessed: 2025-10-14, May 2024.
- [16] N. T. Arun Jothi, S. Anu, K. Harsha, and R. Devi Priya, "USB rubber ducky hunter a proactive defense against malicious USB attacks domain: Cybersecurity," in *2024 International Conference on Intelligent Systems for Cybersecurity (ISCS)*, IEEE, 2024, pp. 1–6.
- [17] Z. Cocoara, *Enterprise USB management and encryption: A comprehensive guide*, en, <https://www.endpointprotector.com/blog/enterprise-usb-management-and-encryption/>, Accessed: 2025-10-14, Dec. 2024.
- [18] S. Gnanavel, M. Sreekrishna, P. Shivaramakrishnan, R. Yaswanth, G. S. Gopika, and S. Supriya, "Key logger for recording the keystroke of the targeted machine," in *2024 International Conference on Computing and Data Science (IC-CDS)*, IEEE, 2024, pp. 1–5.
- [19] S. Khan, C. Devlen, M. Manno, and D. Hou, "Mouse dynamics behavioral biometrics: A survey," en, *ACM Comput. Surv.*, vol. 56, no. 6, pp. 1–33, 2024.
- [20] O. Sen, T. Hassan, A. Ulbig, and M. Henze, "Enhancing SCADA security: Developing a host-based intrusion detection system to safeguard against cyberattacks," 2024. eprint: 2402.14599.
- [21] *What is anomaly detection in Cyber-Security?* en, <https://www.micro.ai/blog/what-is-anomaly-detection-in-cyber-security>, Accessed: 2025-10-14, Feb. 2024.
- [22] A. K. Chillara, P. Saxena, and R. R. Maiti, "Transformer-based GAN-augmented defender for adversarial USB keystroke injection attacks," in *Proceedings of the 26th International Conference on Distributed Computing and Networking*, New York, NY, USA: ACM, 2025, pp. 94–103.
- [23] A. K. Chillara, P. Saxena, and R. R. Maiti, "USB-GATE: USB-based GAN-augmented transformer reinforced defense framework for adversarial keystroke injection attacks," en, *Int. J. Inf. Secur.*, vol. 24, no. 2, 2025.
- [24] D. Ranney, Y. Wang, A. A. Ding, and Y. Fei, "USBSnoop – revealing device activities via USB congestions," 2025. eprint: 2503.03980.
- [25] C.-Y. Wang and F.-H. Hsu, "USBIPS framework: Protecting hosts from malicious USB peripherals," 2025. eprint: 2409.12850.
- [26] <https://www.coro.net/blog/why-usb-attacks-are-back-and-how-to-prevent-them>, Accessed: 2025-10-14.

- [27] https://www.researchgate.net/publication/381031922_To_USBe_or_Not_to_USBe_Discovering_Malicious_USB_Peripherals_through_Neural_Network-Driven_Power_Analysis, Accessed: 2025-10-14.
- [28] <https://www.darkreading.com/vulnerabilities-threats/25-of-malware-spread-via-usb-drives>, Accessed: 2025-10-14.
- [29] <https://venturebeat.com/security/report-9-of-security-incidents-caused-by-usbs-and-other-removable-media>, Accessed: 2025-10-14.
- [30] https://www.researchgate.net/publication/304995447_A_Survey_of_Current_Research_on_CAPTCHA, Accessed: 2025-10-14.
- [31] https://faculty.cc.gatech.edu/~mbailey/publications/oakland18_usb.pdf, Accessed: 2025-10-14.
- [32] https://www.researchgate.net/publication/337952507_USB-Watch_A_Dynamic_Hardware-Assisted_USB_Threat_Detection_Framework, Accessed: 2025-10-14.
- [33] https://www.researchgate.net/publication/331876425_BadUSB_the_threat_hidden_in_ordinary_objects, Accessed: 2025-10-14.
- [34] https://www.researchgate.net/publication/379953124_Enhancing_System_Monitoring_Capabilities_through_the_Implementation_of_Stealthy_Software--Based_Keylogger_A_Technical_Exploration, Accessed: 2025-10-14.
- [35] *AI-powered USB virus alert system for enhanced cybersecurity.*
- [36] *Bot verification*, <https://www.cybersecurityintelligence.com/blog/usb-attacks-the-threat-putting-critical-infrastructure-at-risk-8066.html>, Accessed: 2025-10-14.
- [37] *Cybersecurity in 2024: USB devices continue to pose major threat*, en, <https://www.honeywell.com/us/en/news/2024/04/cybersecurity-in-2024-usb-devices-continue-to-pose-major-threat>, Accessed: 2025-10-14.
- [38] *Is your IoT security strategy scalable?* en, <https://www.iotforall.com/iot-security-scalable>, Accessed: 2025-10-14.
- [39] D. Online, *Versatility unleashed: The diversity of USB accessories in today's tech landscape*, en, <https://www.dqindia.com/business-technologies/versatility-unleashed-the-diversity-of-usb-accessories-in-todays-tech-landscape-3979697>, Accessed: 2025-10-14.
- [40] *Using caution with USB drives*, en, <https://www.cisa.gov/news-events/news/using-caution-usb-drives>, Accessed: 2025-10-14.
- [41] P. Walters, *The risks of using portable devices*, <https://www.cisa.gov/sites/default/files/publications/RisksOfPortableDevices.pdf>, Accessed: 2025-10-14.