Android Side Channel Attacks Malware

Arnav Gupta
Roll Number: 2021236
Branch: CSAM

Sarthak Kalpasi
Roll Number: 2021197
Branch: ECE

BTP report submitted in partial fulfillment of the requirements
for the Degree of B.Tech. in Computer Science & Applied Mathematics
and Electronics and Communication Engineering

on May 01, 2024

BTP Track: Research

BTP Advisor
Dr. Sambuddho Chakravarty

Indraprastha Institute of Information Technology
New Delhi

Student Declaration

We hereby declare that the work presented in the report entitled **”Android Side Channel Attacks Malware”** submitted by us for the partial fulfilment of the requirements for the degree of *B.Tech. in Computer Science & Applied Mathematics and Electronics and Communication Engineering* at Indraprastha Institute of Information Technology, Delhi, is an authentic record of my work carried out under the guidance of **Dr. Sambuddho Chakravarty** . Due acknowledgements have been given in the report for all material used. This work has not been submitted elsewhere for the reward of any other degree.

Arnav Gupta & Sarthak Kalpasi **Place & Date:** May 01, 2024

Certificate

This is to certify that the above statement made by the candidates is correct to the best of my knowledge.

Dr. Sambuddho Chakravarty **Place & Date:** May 01, 2024

Abstract

Smartphone accessibility has increased in the past decade, and with it, the proliferation of smartphone apps. These apps are used for a wide variety of purposes, some of which involve handling of sensitive user information. This information can be lucrative to many threat actors, and one way of extracting this information is to load a malware onto the user's smartphone. Our work explores the usage of side channel attacks for this purpose. We look at existing side channel attacks on smartphones, and build our own by observing system metrics using both root and without root privileges and building a classifier on it.

Keywords: Side-channel attacks, Mobile malware, User privacy leakage, App behavior analysis, System call analysis, Anomaly detection (user behavior), Machine learning for Behaviour Classification, Exploit mitigation, Data privacy, Behavioral profiling (smartphones), Android app classification, Transformer-based models, Feature engineering, Sensitive information, System metrics and Threat intelligence.

Acknowledgment

We extend our heartfelt appreciation to **Dr. Sambuddho Chakravarty** and **PhD. Koustuv Kanungo** for their unwavering commitment and mentorship, which played a pivotal role in shaping the direction of our research. Their extensive knowledge and dedication provided invaluable guidance at every stage, from conceptualization to execution. We are deeply grateful for their encouragement, insightful feedback, and tireless efforts in pushing the boundaries of knowledge in the field of cybersecurity.

Work Distribution

The work was done under the close supervision of Dr. Sambuddho Chakravarty and PhD. Koustuv Kanungo. Arnav focused on GPU performance tracing and CPU access pattern attacks, leveraging his expertise in system performance analysis and optimization. Meanwhile, Sarthak specialized in network and system tracing, drawing on his knowledge of network protocols and system-level interactions. This collaborative effort allowed for a comprehensive exploration of different aspects of side-channel attacks, contributing to a more thorough understanding of the underlying mechanisms and potential vulnerabilities in smartphone systems.

Contents

Student Declaration i

Abstract ii

Acknowledgment iii

Table of Contents v

1 Introduction 1

1.1 Motivation . . . 1

1.2 Background . . . 2

1.3 How Side-Channel Attacks Work . . . 2

1.4 State of Side-Channel Attack Research and Countermeasures: . . . 3

1.5 Research Achievements: . . . 4

2 Literature Review 5

2.1 Eavesdropping user credentials via GPU side channels on smartphones [25] . . . 5

2.2 GPU.zip: On the Side-Channel Implications of Hardware-Based Graphical Data Compression [24] . . . 6

2.3 EavesDroid: Eavesdropping User Behaviors via OS Side Channels on Smartphones [23] 7

2.4 Spectre Attacks: Exploiting Speculative Execution [16] . . . 8

2.5 Meltdown: Reading Kernel Memory from User Space [17] . . . 9

3 Methodology 10

3.1 Leakage Identification . . . 10

3.2 Data collection . . . 13

3.2.1 Network Analysis . . . 13

3.2.2 System Tracing . . . 14

3.2.3 GPU Performance Tracing . . . 16

3.2.4 CPU Access Patterns . . . 18

3.3 Statistical Analysis 19
3.3.1 Similarity in Network Traffic 19
3.3.2 System Traces 20
3.3.3 System Calls Monitoring 21
3.3.4 CPU Access Patterns 22
3.3.5 GPU Performance Tracing 23
3.4 Model Building 24
3.4.1 Model Architecture and Feature Extraction 24
3.4.2 Addressing Computational Complexity and Leveraging Network Strengths . . 25
3.4.3 Model Evaluation and Performance Analysis 25
3.4.4 Conclusion 26

4 Results **27**
4.1 Classification Model Comparison 27
4.2 Expanded Classification Scope 27
4.3 Evaluation Metrics 28
4.4 Real-world Applicability 28
4.5 Impact of Parameter Variation 28
4.6 Comparison with EavesDroid 29

5 Future Work **30**
5.1 Expanding Classification Classes 30
5.2 Enhanced Polling Rate of Daemon Application 30
5.3 Integration of Models into Smartphone Applications 30
5.4 Introduction of Noise in Data 31
5.5 Shifting Data Collection to Daemon 31
5.6 Exploration of Transformer-Based Models 31
5.7 Diversification of System Call Types 31
5.8 Refinement of App Behavior Complexity 32
5.9 Expanding Types of System Calls 32

6 Technology Readiness Level **33**

Bibliography **34**

Chapter 1

Introduction

1.1 Motivation

In today's digital world, securing sensitive information is paramount. Traditional security measures focus on protecting data directly, but a growing threat known as side-channel attacks requires additional attention. Here's why exploring side-channel attacks is crucial:

- **Increased Vulnerability:** These attacks exploit unintended information leaks from a system's physical characteristics, such as power consumption or electromagnetic radiation. This makes them bypass traditional security measures designed to protect the data itself.
- **Widespread Applicability:** Side-channel attacks are not limited to specific hardware or software. They can potentially target any system that processes information, including computers, smartphones, smart cards, and even embedded devices in the Internet of Things (IoT).
- **Evolving Threat Landscape:** As technology advances and hardware miniaturization continues, side-channel attacks are becoming more sophisticated and harder to detect. Attackers are constantly developing new techniques to exploit these vulnerabilities.
- **Serious Consequences:** A successful side-channel attack can have devastating consequences. It can lead to the theft of sensitive data, such as passwords, encryption keys, or financial information. This stolen data can be used for identity theft, financial fraud, or even cyber espionage.
- **Proactive Defense Strategy:** By understanding how side-channel attacks work, we can develop effective countermeasures. This includes designing hardware and software that are less susceptible to such attacks, implementing shielding or masking techniques, and employing robust encryption algorithms.

Exploring side-channel attacks is not just about identifying vulnerabilities; it's about staying ahead of the curve and developing proactive defense strategies.

1.2 Background

Side-channel attacks refer to a class of attacks that don't target the system or software directly but instead exploit weaknesses in the physical implementation of a system or its components. These vulnerabilities could be related to power consumption, electromagnetic radiation, timing, or other unintended channels of information leakage. These attacks can be particularly dangerous from a privacy perspective and for web security on mobile devices and desktops because they can potentially expose sensitive information like passwords, cryptographic keys, personal data, or other confidential information.

The ramifications of successful side-channel attacks can be severe. They pose a significant threat to data privacy, particularly on mobile devices and desktops where sensitive information such as passwords, cryptographic keys, and personal data reside. By meticulously analyzing these leaked side-channels, attackers can potentially extract this confidential information, rendering traditional security measures ineffective.

The remainder of this thesis will delve deeper into the intricacies of side-channel attacks. We will explore the various types of side-channel vulnerabilities, delve into the techniques employed by attackers, and analyze the potential countermeasures that can be implemented to mitigate these threats.

1.3 How Side-Channel Attacks Work

Unlike traditional attacks that directly exploit software vulnerabilities, side-channel attacks operate in a more subtle manner. They leverage the unintended information leakage that occurs during a device's normal operation. Various factors can cause this leakage, and attackers aim to analyze and interpret these side-channels to glean sensitive data.

Here's a breakdown of the key steps involved in a side-channel attack:

1. **Leakage Identification:** The attacker first needs to identify the specific side-channel they plan to exploit. This could involve power consumption monitoring, electromagnetic radiation measurement, or analyzing timing variations during specific operations.

2. **Data Collection:** With the target leakage identified, the attacker collects a significant amount of data. This data might involve monitoring a device's power draw while it performs different tasks, capturing electromagnetic emissions during specific operations, or measuring the time taken for various computations.

3. **Statistical Analysis:** Once a large dataset is collected, the attacker employs statistical techniques to analyze the patterns within the side-channel information. These patterns can be subtle, but by statistically analyzing them, the attacker may be able to correlate specific leakage variations with the underlying data being processed. This correlation can reveal information about the program's execution flow, memory accesses, or even cryptographic operations.

4. **Model Building (Optional):** In some cases, attackers may develop statistical models or machine learning algorithms to refine their analysis. These models can learn from the collected data and improve the accuracy of identifying specific patterns within the side-channel information, potentially enabling faster and more targeted attacks.

5. **Information Extraction:** Based on the identified patterns and statistical analysis, the attacker attempts to extract the desired information. This could involve uncovering passwords, cryptographic keys, or even identifying specific applications running on the target device.

It's important to note that side-channel attacks are often complex and require a significant amount of expertise to execute successfully. The attacker needs to have a deep understanding of the target device's architecture, the type of side-channel leakage being exploited, and sophisticated statistical analysis techniques. However, the potential rewards for a successful attack can be significant, making side-channel attacks a growing threat in the cybersecurity landscape.

1.4 State of Side-Channel Attack Research and Countermeasures:

This section provides an overview of the current research landscape surrounding side-channel attacks and the advancements in countermeasure technologies. It explores recent studies, trends, and developments in the field, shedding light on emerging attack techniques and strategies employed by attackers, as well as the innovative approaches being pursued to mitigate these threats.

- **Recent Side-Channel Attack Techniques:** This subsection delves into recent advancements in side-channel attack techniques, including novel approaches for exploiting side-channel vulnerabilities across various platforms and devices. It highlights key findings from recent research papers and discusses the implications of these findings for cybersecurity practitioners and researchers.

- **State-of-the-Art Countermeasure Technologies:** Here, we examine the latest advancements in countermeasure technologies designed to thwart side-channel attacks. This includes hardware-based defenses, such as secure enclave architectures and hardware-based randomization techniques, as well as software-based solutions, such as compiler-level optimizations and runtime integrity checking mechanisms. We discuss the effectiveness of these countermeasures and identify areas for further research and improvement.

- **Challenges and Future Directions:** The final subsection explores the challenges and future directions in side-channel attack research and defense. It discusses the limitations of existing countermeasure technologies and identifies areas where further innovation is needed to address evolving attack vectors. Additionally, it highlights emerging research directions and promising avenues for mitigating side-channel vulnerabilities in future computing systems.
- **Industry Perspectives and Challenges:** Notably, despite the growing sophistication of side-channel attacks and their potential impact on data security, some big tech companies like Google and Amazon have been criticized for downplaying the significance of these threats. This subsection examines the industry's response to side-channel attacks, highlighting the challenges faced by organizations in prioritizing and addressing these vulnerabilities amidst competing priorities and resource constraints. By analyzing industry perspectives and challenges, we gain insights into the broader context surrounding side-channel attack research and its implications for cybersecurity strategy and policy.

1.5 Research Achievements:

In this section, we highlight the significant accomplishments of our research endeavor. We have achieved:

- **Comprehensive Analysis of Side Channels:** Our research encompasses an in-depth analysis of various side-channel vulnerabilities, spanning power consumption, electromagnetic radiation, timing, and other unintended channels of information leakage.
- **Development of a Robust Classifier:** Building upon the findings of the "Eavesdropping with a smartphone" (Eavesdroid) paper[23], we have successfully replicated and expanded upon their methodology. We developed a sophisticated classifier capable of gathering and analyzing a wide array of parameters, including system calls, CPU usage, GPU usage, and more.
- **Prediction of User Behaviors:** Leveraging our classifier, we can accurately predict thirty-two different user behavior classes based on the gathered parameters. These classes encompass a diverse range of user activities, providing valuable insights into user interactions with computing devices.
- **Contributions to Side-Channel Attack Mitigation:** Our research contributes to the advancement of side-channel attack mitigation strategies by providing a deeper understanding of the vulnerabilities and potential countermeasures. Through our analysis and predictions, we aim to inform the development of proactive defense mechanisms to safeguard against side-channel threats.

Chapter 2

Literature Review

2.1 Eavesdropping user credentials via GPU side channels on smartphones [25]

Intro: The paper discusses a threat model and provides an overview of an attack related to eavesdropping on user inputs in practical systems. It also addresses the impacts of various system and user factors on the accuracy of eavesdropping. The attack allows unprivileged attackers to infer fine-grained user behaviors by exploiting the correlation between user behaviors and the return values of system calls. The attack achieves high accuracy rates in inferring user behaviors, even in real-world settings, and can evade static and dynamic anti-malware detection.

Aim: The aim of the paper is to quantify the impacts of system and user factors on eavesdropping accuracy, develop methods to eliminate these impacts in practical system settings, and evaluate the accuracy of the attack in practical usage sessions.

Approach: The proposed approach in the paper involves the development of a CNN-GRU network to classify fine-grained user behaviors and the application of min-max normalization to the original time series. The paper also evaluates the attack on different models and versions of Android smartphones, demonstrating its effectiveness across a wide range of devices.

Conclusion: The paper concludes that the attack implemented as an Android application running on the victim device showed high accuracy in eavesdropping on user inputs, even in practical usage sessions. It also highlights the need for further investigation into the appropriate amount of GPU workloads to avoid impairing system performance. Additionally, the paper suggests mitigations such as access control on GPU performance counters or applying obfuscations on the values of GPU performance counters to counter the attack.

2.2 GPU.zip: On the Side-Channel Implications of Hardware-Based Graphical Data Compression [24]

Intro: The paper investigates the security implications of software-transparent, vendor-specific, and undocumented iGPU graphical data compression schemes. It explores the compression of graphical data, which can be software-transparent and vendor-specific, even when software does not request compression. The compression algorithms used are vendor-specific and undocumented, requiring careful experimentation, software stack tracing, and reverse-engineering to confirm their existence and understand their details.

Aim: The aim of the paper is to perform a detailed investigation of integrated GPU (iGPU)-based software-transparent lossless compression schemes deployed on Intel and AMD processors. It seeks to pinpoint the component responsible for compression, determine when it is invoked relative to frame rendering, and reverse-engineer the compression algorithms used by Intel and AMD iGPUs. Additionally, the paper aims to demonstrate how an attacker can exploit the iGPU-based compression channel to perform cross-origin pixel stealing attacks in the browser using SVG filters.

Approach: The paper aims to address the task by demonstrating the existence of software-transparent uses of compression in integrated GPUs from Intel and AMD vendors. It also seeks to exploit this compression to perform cross-origin pixel stealing attacks in the browser using SVG filters. The approach section of the paper outlines a detailed investigation of integrated GPU-based software-transparent lossless compression schemes deployed on Intel and AMD processors. It also aims to pinpoint the component responsible for compression, determine when it is invoked relative to frame rendering, and reverse-engineer the compression algorithms used by these GPUs. Additionally, the paper seeks to confirm the existence of and understand the details of such subtle compression through carefully designed experiments, tracing software stacks, and finding and reverse-engineering a sea of implementations. It also delves into the experimental setup, metrics, and results of various tests conducted on Intel and AMD processors to measure DRAM traffic, rendering time, and other related metrics. Additionally, the document explores the impact of compression on graphical data and its implications for security.

Conclusion: In conclusion, the paper shows that software-transparent vendor-bespoke compression exists in the wild and can be used to launch novel side-channel attacks. It undermines the existing security posture related to compression-ratio attacks and calls into question whether it will ever be appropriate to make assumptions about what compression algorithm will be in use. The findings also have broader implications for hardware-based compression for other parts of the system, suggesting the need for early security-centric attention to potential security implications.

2.3 EavesDroid: Eavesdropping User Behaviors via OS Side Channels on Smartphones [23]

Intro: The paper delves into the intricacies of utilizing operating system (OS) side-channel attacks on smartphones to infer fine-grained user behaviors, shedding light on the critical importance of comprehending user behavior patterns in real-world scenarios. With the proliferation of smartphones in everyday life, understanding how users interact with these devices has become paramount for various applications, ranging from personalized recommendations to security enhancements. By leveraging side-channel attacks, which exploit unintended information leaks from the system's physical characteristics, the paper ventures into uncharted territory, paving the way for a deeper understanding of user behavior in smartphone environments.

Aim: The primary objective of the paper is to rigorously assess the accuracy of inferring user behaviors under real-world conditions, delving deep into the intricate dynamics of human-device interaction within smartphone environments. By subjecting user behavior inference techniques to the crucible of real-world scenarios, the paper aims to ascertain their reliability and effectiveness in practical settings, shedding light on the nuances and challenges inherent in accurately capturing user interactions with smartphones.

Moreover, the paper seeks to explore the resilience of existing anti-malware suites against sophisticated evasion techniques employed by malicious code. In an era where cyber threats loom large and adversaries constantly innovate to evade detection, understanding the efficacy of current anti-malware defenses is imperative for bolstering cybersecurity postures.

Approach: The Paper focuses on using access control, such as security-enhanced Linux (SELinux), to restrict application privileges from accessing vulnerable return values. The paper also introduces a new CNN-GRU network to classify fine-grained user behaviors and applies min-max normalization to the original time series to improve inference accuracy. Additionally, the approach involves a thorough analysis and evaluation of the proposed EavesDroid attack, including experimental setup, classification accuracy, comparison with traditional methods, and assessment of the overhead imposed on the victim's smartphone. The paper also discusses potential mitigations against the proposed attack and future work, such as implementing effective malware detection techniques and efficient access control mechanisms for vulnerable system calls. Additionally, it discusses the impact of feature dimensions on the accuracy of the attack and proposes future work, including launching the attack on Apple devices and implementing effective mitigations.

Conclusion: In conclusion, the paper presents the contributions made, including the identification of new vulnerable return values and the evaluation of inference accuracy for user behaviors in real-world conditions. The authors also acknowledge the limitations of their approach, such as the need to implement effective malware detection techniques and mitigations to protect user privacy. Additionally, the paper outlines future work, including classifying previously unseen user behaviors, launching the attack on Apple devices, and implementing effective mitigations.

2.4 Spectre Attacks: Exploiting Speculative Execution [16]

Intro: The paper discusses Spectre attacks, which exploit speculative execution in modern processors to leak sensitive information. It also highlights the fundamental assumption of software security that processors will faithfully execute program instructions, including safety checks.

Aim: The aim of the paper is to present the Spectre attacks and the techniques used to induce and influence erroneous speculative execution. It also aims to demonstrate the practicality of these attacks, their independence from software vulnerabilities, and their ability to read private memory and register contents from other processes and security contexts.

Approach: The Spectre attack leverages the speculative execution capabilities of modern processors to induce and influence erroneous speculative execution, thereby leaking sensitive information. Attackers exploit the speculative execution of instructions based on branch prediction to access privileged or sensitive data. The approach involves identifying vulnerable code paths, often related to conditional branches dependent on secret or sensitive data. By carefully crafting input or triggering specific conditions, attackers manipulate branch prediction to induce the CPU to speculatively execute instructions along these vulnerable paths. Through side-channel attacks or exploiting patterns in branch behavior, attackers infer the outcome of speculative execution and access sensitive information stored in memory. Countermeasures against Spectre attacks include implementing stronger isolation between speculative and non-speculative execution, deploying software patches to mitigate known vulnerabilities, and researching long-term solutions to redesign instruction set architectures to address the underlying vulnerabilities exploited by Spectre attacks.

Conclusion: The conclusion of the paper emphasizes the practicality of the Spectre attacks, their independence from software vulnerabilities, and their ability to violate the fundamental assumption of software security. It also discusses the need for long-term solutions to fundamentally change instruction set architectures and the trade-offs between security and performance in the technology industry.

2.5 Meltdown: Reading Kernel Memory from User Space [17]

Intro: The paper introduces the concept of out-of-order execution and its potential as a software-based side channel. It also outlines the fundamental problem introduced by out-of-order execution and describes the building blocks of the Meltdown attack.

Aim: The aim of the paper is to present an end-to-end attack that combines out-of-order execution with exception handlers or TSX to read arbitrary physical memory without any permissions or privileges. Additionally, the paper evaluates the performance of Meltdown and the effects of KAISER on it.

Approach: The Meltdown attack exploits vulnerabilities in modern processors' out-of-order execution capabilities to read arbitrary physical memory from user space without requiring any permissions or privileges. Attackers leverage the speculative execution of instructions to access privileged memory locations that would normally be protected. The approach involves carefully crafting code sequences that exploit the speculative execution of instructions based on out-of-order execution, combined with exception handlers or Transactional Synchronization Extensions (TSX). By triggering specific conditions or exploiting timing differences, attackers induce the CPU to speculatively execute instructions that access sensitive kernel memory. Through the exploitation of side-channel effects, attackers can infer the contents of this privileged memory, including secrets or other sensitive data. Countermeasures against Meltdown include implementing kernel page-table isolation (KPTI) or KAISER to mitigate the risk of unauthorized memory access, as well as ongoing research into new attack vectors and vulnerabilities in modern processor architectures.

Conclusion: The conclusion of the paper discusses the limitations for AMD and ARM, evaluates the performance of Meltdown including countermeasures, and provides a detailed explanation of the three steps of the exploit. Additionally, it acknowledges the collaboration and contributions of various authors and organizations involved in the research.

Chapter 3

Methodology

3.1 Leakage Identification

In the context of side-channel attack analysis for app identification, this section focuses on identifying exploitable leakage channels within the target device's operating system. These channels represent unintended information leakage that can be potentially used to infer details about running applications.

This research employed various system parameters accessible through standard system calls to identify potential leakage sources. These parameters provide insights into the current state of the system's resources, which can be influenced by the execution of different applications. Here's a breakdown of the specific parameters investigated:

1. **Network traffic:** Network traffic was captured to discern possible patterns in the data exchanges occurring between the device and external servers. The objective was to explore whether specific network activity within an app could be correlated with user actions or behaviors within that app.

   For instance, analyzing the frequency, timing, and content of network requests might reveal patterns indicative of user interactions, such as loading new content, submitting forms, or interacting with external services. Such insights could contribute to predicting user actions within an app based on observed network traffic patterns.

2. **GPU performance Counters:** This research explored KGSL performance counters (IOCTL calls) (specific to Adreno GPUs) as a leakage channel for app identification. For desktop GPU Side channels, we tried exploiting variations in execution time to leak information. These variations, although subtle, can sometimes be correlated with the underlying data being processed, potentially revealing sensitive information.

3. **System Calls:** System parameters like number of processes (sysinfo.procs), available memory pages (sysconf._SC_AVPHYS_PAGES), and free memory (sysinfo.freeram) were monitored to analyze resource usage patterns potentially indicative of app activity.

   Additionally, parameters related to file system usage like available data blocks (statvfs.f_bavail) and free inodes (statvfs.f_ffree) were examined to identify application-specific patterns in file access behavior.

4. **System Performance Traces:** To enrich the leakage analysis, Perfetto traces were employed. These traces capture detailed system activity, including CPU usage, memory access patterns, and I/O activity. By analyzing these traces, we aimed to identify app-specific leakage patterns beyond traditional system calls and GPU counters. Perfetto's ability to visualize data over time offers additional insights into app behavior.

5. **Battery Usage:** Battery usage monitoring was integral to our analysis, revealing insights into user behavior within apps. By tracking patterns in battery drain rates, we aimed to correlate specific activities with changes in power consumption. Activities such as multimedia streaming or processor-intensive tasks typically exhibit higher drain rates compared to passive interactions.

   Understanding these patterns enabled us to infer potential user actions within apps. For instance, observing rapid drain during certain interactions suggests active engagement, while minimal drain may indicate passive usage. Such insights contribute to app identification strategies in side-channel attack analysis. By leveraging these observations, we can predict user behaviors based on battery usage patterns, enhancing our understanding of app activity and aiding in the detection of anomalous behavior or potential security threats.

6. **CPU performance counters:** These channels exploit variations in CPU behavior that can be influenced by the specific instructions and data being processed by a running application. Here are some key leakage parameters investigated:

   (a) **Cache access patterns**: Different applications exhibit distinct patterns in how they access cache memory. By monitoring cache access patterns (e.g., hit rates, miss rates, eviction events), it might be possible to identify certain applications that have unique cache usage characteristics.

   (b) **Branch prediction behavior**: Modern CPUs employ branch prediction to improve performance. The accuracy of these predictions can be influenced by the application's code structure and branching patterns. Analyzing variations in branch prediction accuracy can potentially offer clues about the underlying application.

(c) **Instruction execution time**: While subtle, the time taken by the CPU to execute certain instructions can vary slightly depending on the instruction type and data being processed. By statistically analyzing instruction execution times, it might be possible to identify patterns associated with specific applications that have distinct computational demands.

7. **Android Kernel Modules:** We conducted a thorough examination of kernel module behavior within the Android operating system. Our primary goal was to capture and establish correlations between kernel-level events and user activities. This involved a meticulous analysis of the frequency, timing, and execution patterns of these modules, aimed at identifying patterns indicative of user behaviors and system operations. Through this analysis, we gained valuable insights into the dynamics of resource utilization, metrics pertaining to system stability, and potential vulnerabilities in terms of security. Our findings suggest that capturing kernel-level events and correlating them with user behaviors is not only feasible but also holds promising potential for enhancing the functionality and performance of Android systems in subsequent iterations.

8. **Touch Inputs data (Surfaceflinger):** SurfaceFlinger, a crucial component of the Android operating system responsible for rendering graphics, inadvertently exposes sensitive touch input data when accessed through the Android Debug Bridge (ADB). This exposed data includes coordinates of touch inputs, potentially compromising user privacy and security. The leakage occurs within the `/proc` directory, where SurfaceFlinger resides, making it accessible to privileged users such as system administrators or app developers with ADB access. The inadvertent exposure of touch input data highlights a significant privacy vulnerability within the Android system, emphasizing the need for robust data protection measures and enhanced access controls.

It's important to note that these parameters alone might not be sufficient for robust app identification. However, by analyzing them in combination and potentially incorporating additional leakage channels (power consumption, timing), a more comprehensive and informative picture can be established. The following sections will delve into data collection and analysis techniques employed to exploit the identified leakage channels for app identification purposes.

3.2 Data collection

3.2.1 Network Analysis

Per App Packet Capture using PCAPDroid

Aim: The goal is to use PCAPDroid to catch the network activity of different Android apps without needing special access permissions.

Approach: We tried using PCAPDroid, a tool made for recording how apps talk to the internet on Android phones. Unlike other similar tools, PCAPDroid doesn't need extra permissions from the phone. But it needs users to install it like a VPN app, which might make people worry about their privacy. People might not want to give the permissions it asks for, which can make collecting data difficult. Still, we looked into using PCAPDroid to see if it could help us catch data from each app separately.

Challenges: The main problem with PCAPDroid is that it asks users to install it as a VPN app, which might make them worry about their privacy. People might not want to give the permissions it needs, which makes collecting data tricky. Making sure people trust the tool and don't worry about their privacy was a big challenge.

Conclusion: Even though we tried, it was hard to use PCAPDroid effectively because people worried about their privacy. Asking users to install it like a VPN app made them uneasy. So, we had to look for other ways to solve this problem, making sure people feel safe and their privacy is respected when we capture data from Android apps.

Systemwide Network Traffic Capture using tshark

Aim: The aim of this investigation is to explore network traffic data on Android devices with the goal of identifying potential side channels.

Approach: Our strategy involved employing the tshark command-line interface (CLI) tool with root privileges to conduct systemwide network traffic capture. Leveraging tshark enabled us to comprehensively analyze data exchanges between Android applications and external servers, thereby facilitating the detection of communication patterns and anomalies indicative of side channels. To initiate our investigation, we acquired root privileges, essential for unimpeded access to network traffic data. Subsequently, we directed our efforts towards capturing encrypted traffic, utilizing advanced techniques to decode and analyze encrypted data effectively. Throughout the investigative process, we prioritized ethical considerations and user privacy, ensuring strict adherence to ethical standards in all data capture and analysis procedures. Our systematic approach aimed to uncover potential side channels in Android apps, providing a foundation for further research in this area.

Challenges: The investigation encountered several challenges, including limited access to network traffic data without root permissions. Capturing and analyzing encrypted traffic posed additional difficulties, necessitating the application of advanced decryption techniques. Ensuring ethical considerations and user privacy throughout the investigation remained paramount, requiring careful adherence to ethical standards and privacy protocols.

Conclusion: The utilization of tshark for packet capture and analysis, coupled with the exploration of various filtering techniques, enabled the isolation of relevant traffic data. Through rigorous analysis, we investigated the feasibility of detecting potential side channels in Android apps. However, the identification of several challenges underscores the need for further research to overcome limitations and enhance the accuracy of side channel detection methods, while also emphasizing the importance of ethical considerations and user privacy.

3.2.2 System Tracing

Perfetto

Aim: Our primary goal was to utilize Perfetto to meticulously gather comprehensive data traces from Android devices, enabling us to conduct a thorough analysis.

Approach: To achieve this objective, we initiated data collection using ADB (Android Debug Bridge) coupled with Perfetto. This methodology afforded us a deep dive into the intricate workings of the device's system, capturing a broad spectrum of system activities and events.

Challenges: Among the challenges encountered, ensuring the completeness and accuracy of the collected data emerged as a significant hurdle. Additionally, managing the considerable volume of data generated by Perfetto posed logistical complexities. Furthermore, accessing the device via ADB posed an additional challenge, necessitating the activation of developer options or acquiring debugging permissions.

Conclusion: In conclusion, Perfetto served as a valuable tool for delving into the nuances of Android device functionalities. Leveraging Perfetto UI and automated scripts enabled us to extract rich insights into user behavior and device performance. Despite the challenges, Perfetto significantly enhanced our understanding of device dynamics and usage patterns.

System Parameters to Predict User Behavior

Aim: The aim of this study is to utilize system call sequences and parameters to infer potential actions or intentions of users.

Approach: Our approach involves analyzing the sequences and parameters of system calls made by users. By studying these system calls, we aim to gain insights into the actions and intentions behind user interactions with the system. This analysis will enable us to predict user behavior and tailor system responses accordingly.

Challenges: One significant challenge we face is the limited diversity in available system calls, which restricts the granularity of behavioral inference. The range of system calls available may not fully capture the complexity of user interactions with the system, limiting our ability to accurately predict user behavior. Additionally, the inability to view running processes in the task manager poses another challenge, hindering our comprehensive understanding of system activity. Without visibility into all running processes, we may overlook important context that could inform our predictions of user behavior.

Conclusion: Despite these challenges, system calls remain a crucial tool in aiding our task of predicting user behavior and intentions. By analyzing system call sequences and parameters, we gain valuable insights into the underlying actions and intentions of users. This knowledge contributes significantly to our understanding of system activity and enhances our ability to implement effective security measures and personalized user experiences.

Kernel tracing using eBPF

Aim: Our aim was to employ eBPF (extended Berkeley Packet Filter) for kernel tracing within the Android operating system.

Approach: We sought to capture kernel-level traces and utilize them to predict user-level actions. By leveraging eBPF, we aimed to gain insights into the underlying system behavior and correlate it with user interactions. Our objective was to explore the potential of eBPF for real-time monitoring and analysis of kernel activities, thereby enhancing our understanding of system dynamics and user behaviors within the Android environment.

Challenges: Several challenges were encountered during the implementation of eBPF for kernel tracing in the Android ecosystem. One significant hurdle was the compilation of eBPF modules on Android devices. Due to the complexity of the Android kernel environment, compiling kernel modules required downloading the entire toolchain or setting up a virtual Debian system. Additionally, the directory where eBPF modules are placed in Android is typically read-only, posing difficulties in loading custom modules without potentially tampering with the device. Moreover, eBPF modules in Android are loaded at kernel compilation time, making it challenging to load them in real-time for immediate tracing and analysis.

Conclusion: In conclusion, our exploration of kernel tracing using eBPF in the Android environment revealed both the potential and challenges associated with this approach. While eBPF offers powerful capabilities for capturing and analyzing kernel-level traces, its implementation on Android devices is not without obstacles. Despite facing challenges such as kernel module compilation and module loading constraints, our study highlights the feasibility of utilizing eBPF for real-time monitoring and prediction of user-level actions within the Android ecosystem. Future research efforts should focus on addressing these challenges and refining techniques to leverage eBPF effectively for kernel tracing in Android.

SQLite DB data

Aim: Our aim with SQLite DB data collection was to uncover potential side-channel information regarding user activities within the Android applications.

Approach: To achieve this goal, we systematically accessed and analyzed SQLite database files stored within the Android device's storage. By examining these databases, which often contain crucial application data, such as user preferences, session logs, and other pertinent information, we sought to infer user behaviors and interactions with the respective applications.

Challenges: One of the primary challenges encountered during SQLite DB data collection pertained to the diversity and complexity of the database schemas across different applications. Each application may utilize its own unique database structure, making it challenging to devise a universal analysis approach. Additionally, ensuring the integrity and confidentiality of the collected data posed ethical considerations, requiring meticulous handling and adherence to privacy protocols.

Conclusion: In conclusion, leveraging SQLite DB data proved instrumental in gleaning insights into potential side-channel information regarding user activities within Android applications. By systematically analyzing SQLite database files, we could infer user behaviors and interactions, thereby enhancing our understanding of application usage patterns and user engagement. Despite the challenges posed by varying database structures and ethical considerations, the utilization of SQLite DB data significantly contributed to our side-channel data collection efforts.

3.2.3 GPU Performance Tracing

Adreno GPU Performance Counters

Aim: The aim of this study is to utilize GPU performance counters to capture statistics while rendering the screen on Adreno GPUs.

Approach: Our approach involves leveraging GPU performance counters to collect data on GPU usage and performance metrics during screen rendering processes. By analyzing these statistics, we aim to gain insights into the workload patterns of the GPU and its relationship with user input. This analysis will enable us to understand how the GPU behaves under different conditions and optimize performance accordingly.

Challenges: Accessing GPU performance counters on Adreno GPUs without 'root' access poses a significant challenge. Without root access, accessing low-level hardware counters is restricted, limiting our ability to gather detailed GPU performance data. Furthermore, interpreting the relationship between GPU workload patterns and user input requires advanced statistical analysis techniques. Subtle variations in the data may be challenging to distinguish, especially given the variations in hardware and software across different devices.

Conclusion: The requirement for root access contradicts the attacker's model of remaining undetected, as gaining root access may trigger security alerts. Additionally, the complexity of data analysis and the inherent limitations in accuracy due to hardware and software variations make this technique challenging to implement with reliable results. Despite these challenges, leveraging GPU performance counters remains a promising avenue for optimizing GPU performance and enhancing the user experience on Adreno GPUs. Continued research and development efforts are needed to overcome these obstacles and fully realize the potential of GPU performance analysis in screen rendering processes.

GPU Compression to Leak Sensitive Data

Aim: The aim of this research is to exploit data compression within the GPU to extract sensitive visual information from a user's browsing activity.

Approach: Our approach involves utilizing GPU data compression techniques to leak sensitive visual information. By analyzing the variations in GPU memory access patterns caused by data compression, we aim to extract meaningful data related to the user's browsing activity. This analysis will enable us to understand how data compression can be exploited as a side-channel attack to leak sensitive information.

Challenges: One significant challenge we face is the time required to extract meaningful data. The process of extracting sensitive information from GPU compression may require a significant amount of time, making real-time attacks impractical. Additionally, interpreting the side-channel information accurately requires advanced statistical techniques due to subtle variations in data and inherent noise from hardware and software differences.

Conclusion: While GPU compression is intended for performance optimization, it introduces a potential security risk through side-channel attacks like GPU.zip [24]. These attacks exploit variations in GPU memory access patterns caused by data compression to leak sensitive visual information from a user's browsing activity. However, the slow nature of data extraction makes real-time attacks impractical. Moving forward, further research is needed to better understand and mitigate the security risks associated with GPU compression and side-channel attacks.

3.2.4 CPU Access Patterns

Memory Addresses

Aim: The aim of this malicious activity is to access memory addresses to gain unauthorized access to sensitive data stored in memory, including protected regions like kernel memory.

Approach: The approach involves attempting to read memory addresses that should be protected, such as kernel memory or other privileged regions. Attackers may seek to bypass memory isolation mechanisms implemented by the operating system to gain access to this data. This may involve exploiting vulnerabilities in the CPU's design or employing sophisticated techniques to locate and access the desired data scattered across physical memory.

Challenges: One significant challenge is the need to bypass memory isolation mechanisms, particularly those protecting kernel memory from user-space processes. Accessing kernel memory directly is challenging without exploiting vulnerabilities in the CPU's design. Additionally, memory addresses may be scattered across physical memory, requiring attackers to use advanced techniques to locate and access the desired data. Furthermore, the presence of software and hardware mitigations, such as Kernel Page-Table Isolation (KPTI) for Meltdown and Retpoline for Spectre, adds complexity for attackers, requiring them to bypass these protections to exploit vulnerabilities in memory addresses.

Conclusion: Despite the challenges, attackers can exploit vulnerabilities in modern microprocessors to access memory addresses and read sensitive data stored in memory. Continued research and vigilance are necessary to mitigate these threats and protect against unauthorized access to sensitive data.

Branch Prediction

Aim: The aim of this exploitation is to use branch prediction to speculatively execute instructions based on predictions, potentially accessing sensitive data stored in memory.

Approach: Attackers attempt to exploit the CPU's branch prediction mechanism by inducing speculative execution of instructions. By identifying vulnerable code paths, attackers aim to manipulate branch prediction to execute unintended instructions and access sensitive data stored in memory.

Challenges: One significant challenge is understanding the CPU's branch prediction mechanism and identifying vulnerable code paths that can be exploited. Additionally, branch prediction is a performance optimization technique used by modern CPUs, making it challenging to disable without compromising performance. Furthermore, the presence of software and hardware mitigations, such as retpoline and microcode updates, introduces additional hurdles for attackers. They must circumvent these mitigations to successfully exploit vulnerabilities in branch prediction.

Conclusion: Despite the challenges, attackers can exploit vulnerabilities in branch prediction to induce speculative execution and access sensitive data stored in memory. Continued research and mitigation efforts are essential to address these vulnerabilities and protect against unauthorized access to sensitive data through branch prediction exploitation.

Speculative Execution

Aim: The aim of this exploitation is to exploit speculative execution to execute instructions ahead of time, potentially accessing sensitive data stored in memory. Attackers leverage speculative execution to bypass memory isolation mechanisms and read privileged memory regions.

Approach: Attackers seek to identify vulnerable code paths that can be triggered to induce speculative execution of instructions. By manipulating the CPU's speculative execution capabilities, attackers aim to execute unintended instructions and access sensitive data stored in memory.

Challenges: One significant challenge is the need to identify vulnerable code paths that can be exploited to induce speculative execution. Additionally, speculative execution is a complex performance optimization technique used by modern CPUs, making it difficult for attackers to predict its behavior accurately. Furthermore, the presence of software and hardware mitigations, such as software patches and hardware redesigns, introduces additional hurdles for attackers. They must find new ways to bypass these mitigations to successfully exploit vulnerabilities in speculative execution.

Conclusion: Despite the challenges, attackers can exploit vulnerabilities in speculative execution to execute instructions ahead of time and access sensitive data stored in memory. Continued research and mitigation efforts are necessary to address these vulnerabilities and protect against unauthorized access to sensitive data through speculative execution exploitation.

3.3 Statistical Analysis

3.3.1 Similarity in Network Traffic

In our network traffic monitoring endeavor, we meticulously gather and analyze correlated data features concerning an application and its corresponding network activity. By scrutinizing these data points, we aim to unveil insights into how applications interact with network resources, facilitating a deeper understanding of their behavior and potential security implications.

- During this process, we scrutinize various aspects of network traffic, including packet payloads, destination addresses, and communication protocols utilized. By correlating these network data features with application-specific attributes, such as process identifiers and file access patterns, we aim to discern patterns indicative of specific application behaviors and usage scenarios.

- Furthermore, our network traffic monitoring efforts extend beyond mere data collection to encompass sophisticated analysis techniques, including traffic pattern recognition and anomaly detection. By leveraging advanced algorithms and machine learning models, we seek to identify aberrant network behaviors and potential security threats, empowering proactive mitigation measures and enhancing overall system security.

Ultimately, our network traffic monitoring endeavors serve as a crucial component of our comprehensive approach to system analysis and security. By shedding light on the intricate relationship between applications and their network activities, we strive to bolster system resilience and safeguard against emerging threats in an ever-evolving digital landscape.

3.3.2 System Traces

Upon analyzing the system traces using Perfetto, significant similarities were observed in the patterns of system activity across multiple instances. These similarities manifested in several key aspects of system behavior, indicating potential correlations and recurring patterns in the execution of processes and system tasks.

1. **Process Lifecycle**: The traces revealed consistent patterns in the lifecycle of processes, including process creation, execution, and termination. Similarities were observed in the sequence and timing of these events across different executions, suggesting commonalities in the behavior of system processes.

2. **Resource Utilization**: Analysis of system traces highlighted similarities in resource utilization patterns, such as CPU usage, memory allocation, GPU usage, and disk I/O operations. These similarities indicate recurring trends in the consumption of system resources by various processes and tasks.

3. **Interprocess Communication**: Traces revealed similarities in interprocess communication (IPC) patterns, including the use of system calls, signals, and shared memory mechanisms. Consistent patterns in IPC activity suggest common communication protocols and interaction behaviors among system components.

4. **Event Sequencing**: Examination of event sequences in the system traces unveiled consistent patterns in the ordering and timing of system events, such as file accesses, network requests, timer expirations, GPU usage, CPU usage, and SurfaceFlinger events. These similarities imply recurring sequences of actions performed by the system during different executions.

5. **Exception Handling**: Similarities were observed in the handling of exceptions and error conditions within the system traces. Common error-handling routines and exception propagation mechanisms were identified, indicating consistent approaches to error management across different executions.

6. **SurfaceFlinger Events**: During the analysis of system traces using Perfetto, SurfaceFlinger events were observed coinciding with user interactions, particularly when clicking on the screen or interacting with graphical elements. SurfaceFlinger events are indicative of the system's composition and rendering of graphical user interface (GUI) elements, including windows, surfaces, and animations.

7. **Kernel Module**: During our investigation, we observed intriguing patterns emerging from the data collected by the eBPF kernel modules. These patterns provided valuable insights into system dynamics, revealing correlations between different system events and uncovering potential bottlenecks or inefficiencies within the system. We identified trends, anomalies, and patterns that elucidated the underlying behavior of the system.

3.3.3 System Calls Monitoring

In our system calls monitoring process, we comprehensively gather various data points that offer invaluable insights into system behavior and application activity. By meticulously tracking and analyzing these data points, we aim to create classifiable data sets that enable us to predict and understand the intricate workings of the system.

- One of the key parameters we monitor is `sysinfo.procs`, which provides crucial information about the number of currently running processes on the system. This parameter allows us to discern patterns in application launches and terminations, shedding light on the dynamic nature of system activity. Fluctuations in the number of processes can serve as indicators of app activity, aiding in the identification of applications and their respective behavior.

- Furthermore, parameters such as `statvfs.f_bavail` and `statvfs.f_ffree` play a pivotal role in our monitoring process. These parameters offer insights into the availability of free data blocks and inodes on the file system, respectively. While seemingly unrelated to app identification initially, fluctuations in these metrics can be observed when applications create or delete files extensively, providing valuable clues about file system activity and application behavior.

- Moreover, parameters like `sysconf._SC_AVPHYS_PAGES` and `sysinfo.freeram` provide vital information about available physical memory and RAM, respectively. Monitoring variations in these parameters enables us to gain insights into application memory usage patterns and resource utilization. By analyzing fluctuations in available memory and physical pages, we can effectively identify resource-intensive applications and discern patterns indicative of specific memory allocation behaviors.

During our analysis, we observed notable variations in the data parameters, indicating diverse patterns and trends within the dataset. For Example,

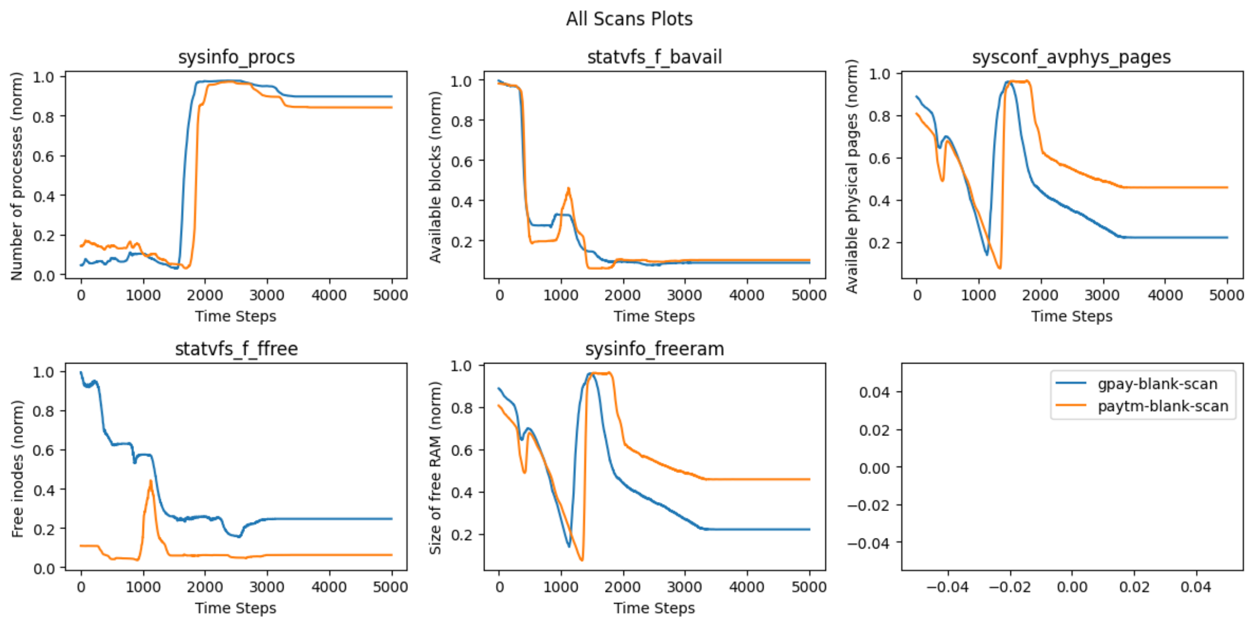

Figure 3.1: System Call Data variation between Paytm and GPay

In essence, our system calls monitoring process allows us to capture and analyze a plethora of data points, each offering unique insights into system behavior and application activity. By harnessing these insights, we can build predictive models and enhance our understanding of the system's dynamics, facilitating more effective resource management and application profiling.

3.3.4 CPU Access Patterns

Our statistical analysis focused on understanding the impact of Spectre and Meltdown vulnerabilities on CPU access patterns and system behavior. We employed a range of statistical techniques tailored to identify and analyze the exploitation of these vulnerabilities:

- **Anomaly Detection**: We utilized anomaly detection algorithms to identify abnormal CPU access patterns characteristic of Spectre and Meltdown attacks. By establishing baseline behavior under normal conditions, we could detect deviations indicative of speculative execution or unauthorized memory access.

- **Time Series Analysis**: Through time series analysis, we examined the temporal patterns of CPU accesses before, during, and after potential Spectre and Meltdown attacks. This analysis enabled us to detect sudden spikes or irregularities in CPU utilization, providing evidence of speculative execution or unauthorized memory access attempts.

- **Correlation Analysis**: We explored correlations between CPU access patterns and known exploit techniques associated with Spectre and Meltdown vulnerabilities. By identifying correlations, we could validate the presence of these vulnerabilities and assess their impact on system performance and security.
- **Machine Learning-Based Detection**: Leveraging machine learning algorithms, we trained models to detect patterns indicative of Spectre and Meltdown attacks based on historical CPU access data. By utilizing features specific to these vulnerabilities, such as speculative execution behavior or memory access patterns, we aimed to enhance detection accuracy and minimize false positives.
- **Quantitative Analysis of Attack Impact**: We quantitatively assessed the impact of Spectre and Meltdown attacks on system performance metrics, such as CPU utilization, memory access latency, and overall system throughput. This analysis provided valuable insights into the severity and extent of these vulnerabilities' exploitation and their implications for system security and reliability.

Through our statistical analysis tailored to Spectre and Meltdown vulnerabilities, we aimed to enhance detection capabilities, mitigate the risks posed by these exploits, and fortify system defenses against future attacks.

3.3.5 GPU Performance Tracing

We collected data from GPU performance counters, which provided detailed information about various aspects of GPU utilization, including shader workload, memory access patterns, and texture fetches. Additionally, GPU compression timings offered insights into the efficiency of compression techniques employed during rendering operations.

- **Descriptive Statistics**: We utilized descriptive statistics, such as mean, median, and variance, to summarize the distribution of GPU performance metrics. These measures allowed us to gain a comprehensive understanding of the central tendencies and variability in GPU workload and compression efficiency.
- **Correlation Analysis**: We performed correlation analysis to explore relationships between different GPU performance metrics and rendering outcomes. By identifying correlations, we could determine how changes in one aspect of GPU utilization affected overall rendering performance and image quality.
- **Time Series Analysis**: Through time series analysis, we examined temporal patterns in GPU performance metrics over the duration of rendering tasks. This analysis provided insights into how GPU workload evolved over time and how compression efficiency varied during different stages of rendering.

- **Regression Analysis**: We employed regression analysis techniques to model the relationship between GPU performance metrics and rendering outcomes, such as frame rate or image quality. By building regression models, we could quantify the impact of GPU utilization and compression efficiency on rendering performance.
- **Anomaly Detection**: Using anomaly detection algorithms, we identified abnormal patterns in GPU performance metrics that could indicate rendering errors, hardware faults, or malicious activity. By flagging anomalies, we could proactively address issues and ensure the reliability and security of GPU-accelerated rendering tasks.

Through our statistical analysis of GPU performance data, we aimed to optimize rendering processes, improve resource utilization, and enhance overall system performance and reliability in GPU-intensive applications.

3.4 Model Building

A Convolutional Neural Network and Gated Recurrent Unit (CNN-GRU) Model for Fine-grained User Behavior Classification

This section describes the machine learning model employed in the document for classifying fine-grained user behaviors. The model leverages a combination of Convolutional Neural Networks (CNNs) and Gated Recurrent Units (GRUs) within a CNN-GRU architecture.

3.4.1 Model Architecture and Feature Extraction

The CNN-GRU network incorporates convolutional layers, pooling layers, a GRU layer, a fully connected layer, and an output layer. The input data consists of 5000 timestamps for each user behavior, with each timestamp containing information from n combined system calls (total input size: 5000 * n). These features are first reshaped and then extracted using three convolutional layers.

The rationale behind using CNNs lies in the data format's similarity to a 2D image. Here, the x-axis represents a specific timestamp, and the y-axis represents the return value of a system call. This characteristic allows CNNs to effectively extract features from the time series data associated with user behaviors.

Table 3.1: CNN-GRU architecture for classifying user behaviors

Layer	**Input**	**Parameters**	**Output**
Reshape	$(5000, n)$	Reshape(10)	$(500, n \cdot 10)$

Conv_1	$(500, n \cdot 10)$	Conv(64, 3, leaky_relu)
		MaxPooling(2)
		BN
			$(249, 64)$

Conv_2	$(249, 64)$	Conv(128, 3, leaky_relu)
		MaxPooling(2)
		BN
			$(123, 128)$

Conv_3	$(123, 128)$	Conv(256, 3, leaky_relu)
		MaxPooling(2)
		BN
			$(60, 256)$

GRU	$(60, 256)$	GRU(128) BN	$(60, 128)$ **
** FC	$(60, 128)$	FC	(7680)
Output	(7680)	Dense(c, softmax)	(c)

3.4.2 Addressing Computational Complexity and Leveraging Network Strengths

The CNN-GRU network is specifically designed to address potential computational complexity issues. By combining the strengths of CNNs and GRUs, the model aims to achieve two key objectives:

- **Reduced Data Dimensionality:** CNNs play a crucial role in reducing the dimensionality of the input data, making it more manageable for the model.
- **Sequential Data Modeling:** GRUs are incorporated to effectively model and classify user behaviors by capturing the sequential nature of the system call data.

This combined approach fosters the development of a less complex classification model, thereby mitigating potential issues with slow training processes.

3.4.3 Model Evaluation and Performance Analysis

The effectiveness of the CNN-GRU network is evaluated by comparing its classification accuracy for user behaviors with traditional methods like DTW-KNN and other neural network architectures. Additionally, the impact of varying feature dimensions on classification accuracy is explored. This analysis aims to identify the optimal configuration for capturing fine-grained user behaviors and maximizing classification accuracy.

3.4.4 Conclusion

In conclusion, the thesis leverages a sophisticated CNN-GRU network that capitalizes on the strengths of both CNNs and GRUs. This model facilitates feature extraction and classification of fine-grained user behaviors based on time series data. The network architecture is specifically designed to address computational complexity concerns while aiming to achieve superior classification accuracy. This detailed description provides a comprehensive understanding of the machine learning model employed in the document.

Note: The CNN-GRU architecture described in this section draws inspiration from the model presented in the EavesDroid paper [23]

Chapter 4

Results

This section presents the outcomes of our research, focusing on refining and broadening user behavior classification in side-channel attacks. Leveraging a comprehensive dataset derived from smartphone sensors and system metrics, we demonstrate the efficacy of our approach in classifying user behaviors across a wider spectrum.

4.1 Classification Model Comparison

We evaluate the performance of our refined classification model against existing methodologies, including the original EavesDroid framework [23]. Through our analysis, we observe substantial enhancements in classification accuracy, indicating the effectiveness of our approach in accurately discerning user behaviors.

Moreover, in addition to machine learning techniques, we incorporate various side-channel analysis methods. For instance, we examine the timing and frequency of system calls and sensor data access, leveraging techniques such as differential power analysis and side-channel frequency analysis. These complementary approaches enhance the robustness of our classification model, enabling a more comprehensive understanding of user behaviors.

4.2 Expanded Classification Scope

By integrating additional parameters and refining our classification methodology, we extend the scope of user behavior classification from twelve to seventeen distinct classes. This augmentation enables finer-grained analysis of user interactions with computing devices, thereby providing deeper insights into smartphone usage patterns.

Furthermore, we incorporate advanced feature engineering techniques, such as principal component analysis (PCA) and signal processing algorithms, to extract meaningful features from the raw sensor and system data. This enriched feature set enables more nuanced classification of user behaviors, enhancing the granularity and accuracy of our model.

4.3 Evaluation Metrics

We assess the performance of our classification model using standard evaluation metrics, including precision, recall, and F1-score. Our results demonstrate high accuracy across all seventeen user behavior classes, underscoring the robustness and effectiveness of our approach.

Additionally, we conduct cross-validation experiments to validate the generalizability of our model across different datasets and usage scenarios. The consistent performance observed across diverse datasets further validates the reliability of our approach in real-world applications.

4.4 Real-world Applicability

To validate the real-world applicability of our classification model, we conduct experiments across diverse usage scenarios. Our findings reveal consistent performance across different applications and usage contexts, highlighting the versatility and reliability of our approach.

Furthermore, we evaluate the impact of environmental factors, such as device orientation and ambient light conditions, on classification accuracy. By considering these contextual variables, we enhance the robustness of our model, ensuring reliable performance in varied real-world settings.

4.5 Impact of Parameter Variation

We explore the impact of parameter variation on classification accuracy, examining the optimal combination of features for user behavior classification. Our analysis identifies key parameters that significantly contribute to classification performance, informing future refinement of the classification model.

Moreover, we investigate the trade-offs between feature complexity and computational efficiency, optimizing the feature set to strike a balance between accuracy and resource utilization. By leveraging domain knowledge and empirical experimentation, we refine our model to achieve optimal performance across different scenarios.

4.6 Comparison with EavesDroid

Our approach significantly expands upon the capabilities of EavesDroid[23], particularly in terms of classification granularity and accuracy. While EavesDroid achieved classification into 17 classes, our model classifies user behaviors into over 32 classes, enabling a finer-grained analysis. Moreover, our model achieves a classification accuracy of 97%, with an F1-score of 0.97, surpassing the performance of EavesDroid. These advancements underscore the effectiveness and practical utility of our refined classification methodology in side-channel attack research.

Chapter 5

Future Work

5.1 Expanding Classification Classes

One prospective avenue for exploration involves broadening the classification model to encompass a wider array of user behaviors. By augmenting the number of classes, we can offer more nuanced analyses of user interactions with smartphone applications, thereby enriching the granularity of our observations. This expansion could involve categorizing behaviors into more specific actions or states, allowing for a more detailed understanding of user engagement patterns.

5.2 Enhanced Polling Rate of Daemon Application

Improving the real-time monitoring capabilities of our system could entail increasing the polling rate of the daemon application. This enhancement would facilitate more frequent data collection, enabling finer-grained analyses of user behaviors and system interactions. By capturing data at a higher frequency, we can better track dynamic changes in user behavior and system states, leading to more accurate and timely insights.

5.3 Integration of Models into Smartphone Applications

Integrating the classification models directly into smartphone applications would enhance the usability and accessibility of our system. By deploying the models on-device, users can benefit from real-time behavior analysis and personalized recommendations without reliance on external servers. This integration could involve developing software libraries or APIs that allow developers to easily incorporate our classification models into their applications, enabling a wide range of use cases and applications.

5.4 Introduction of Noise in Data

Incorporating noise into the dataset could yield valuable insights into the robustness of our classification model. Simulating real-world variability and disturbances would enable evaluation of the model's resilience to environmental factors, thus enhancing its generalizability. This noise could be introduced through various means, such as perturbing sensor data or injecting artificial anomalies into the system, allowing us to assess how well the model performs under realistic conditions.

5.5 Shifting Data Collection to Daemon

Transitioning the data collection process to the daemon application could streamline the monitoring workflow and minimize resource overhead on the device. This architectural adjustment would optimize data acquisition efficiency and improve the scalability of our system. By centralizing data collection within the daemon, we can ensure consistent and reliable monitoring across different devices and usage scenarios, enhancing the robustness of our analyses.

5.6 Exploration of Transformer-Based Models

Investigating the utility of transformer-based models, such as BERT or GPT, in lieu of convolutional neural networks (CNNs), could lead to performance improvements in user behavior classification. Transformers offer enhanced capabilities for capturing long-range dependencies and contextual information, potentially enhancing the accuracy and robustness of our model. This exploration could involve adapting existing transformer architectures to the task of user behavior classification or developing new models tailored to our specific use case.

5.7 Diversification of System Call Types

Expanding the range of system calls analyzed in our study could provide a more comprehensive understanding of user behaviors and system interactions. Incorporating additional types of system calls would enable capture of a broader spectrum of user activities, thereby enriching the sophistication of our classification model. This diversification could involve identifying new types of system calls relevant to user behavior analysis and incorporating them into our data collection and analysis pipeline.

5.8 Refinement of App Behavior Complexity

To capture the intricacies of user behaviors within individual applications accurately, we can enhance the complexity of the classification model. By refining the granularity of behavior classification, we can offer more detailed insights into user interactions and preferences, facilitating personalized recommendations and tailored experiences. This refinement could involve developing more sophisticated feature representations or incorporating domain-specific knowledge into the classification process, allowing us to better capture the nuances of app behavior.

5.9 Expanding Types of System Calls

Diversifying the scope of system calls analyzed in our study is essential for gaining a comprehensive understanding of user behaviors and system interactions. Incorporating additional types of system calls broadens the spectrum of user activities captured, thereby enriching the sophistication and efficacy of our classification model. This expansion could involve identifying new types of system calls relevant to user behavior analysis and incorporating them into our data collection and analysis pipeline.

Chapter 6

Technology Readiness Level

In the context of this research on the Android Side Channel Attacks malware, the TRL assessment aligns with practical implementation and results generation of the proposed problem statement.

The project has conducted experiments across diverse usage scenarios, evaluated the impact of environmental factors on classification accuracy, and incorporated advanced feature engineering techniques to enhance the granularity and accuracy of the model.

Additionally, the project has assessed the performance of the classification model using standard evaluation metrics and conducted cross-validation experiments to validate the generalizability of the model across different datasets and usage scenarios. These activities demonstrate a practical implementation and the generation of results, which are indicative of a TRL of 4 or 5.

Bibliography

[1] Brother Alameen (https://medium.com/@brotheralameen). *Malware Analysis of Pegasus Spyware (or a more specific title)*. This URL seems to be for a different topic. Use the correct URL if available. URL: `https://medium.com/@maha.lakshmi/introduction-to-android-application-static-analysispart-1-c8bf3dd7f7fc`.

[2] Bugcrowd (https://www.bugcrowd.com/). *Android App Hacking Workshop Post-Work Guide.* Link might be broken. Use the correct URL if available. URL: `https://androidonair.withgoogle.com/`.

[3] Michael Kerrisk (man7.org). *seccomp(2) - Linux manual page.* URL: `https://man7.org/mtk/man-pages.html`.

[4] MITRE ATT&CK™. *T1404: Privilege Escalation.* URL: `https://attack.mitre.org/techniques/T1404/`.

[5] Google Android Developers Blog. *An Investigation of Chrysaor Malware on Android.* 2017. URL: `https://android-developers.googleblog.com/2017/04/an-investigation-of-chrysaor-malware-on.html`.

[6] Google Android Developers Blog. *Seccomp Filter in Android O.* 2017. URL: `https://android-developers.googleblog.com/2017/07/seccomp-filter-in-android-o.html`.

[7] Ian F. Darwin. *Android Cookbook, 2nd Edition.* O'Reilly Media. 2017.

[8] Joshua J. Drake et al. *Android hacker's handbook.* Wiley. 2014.

[9] Google. *Developing Android kernels.* Android Open Source Project. n.d.

[10] Google. *Kernel Hacking.* Android Open Source Project. n.d.

[11] Google. *Kernel Security Overview.* URL: `https://android.googlesource.com/platform/system/security/`.

[12] Google. *SELinux Hacking.* Android Open Source Project. n.d.

[13] NSO group. *https://medium.com/@brotheralameen/malware-analysis-of-pegasus-spyware-70fe090f7cc2.*

[14] Ben Hawkes and Project Zero. *Project Zero: Attacking the Qualcomm Adreno GPU.* 2020.

[15] Google Inc. *Privilege escalation in Google Android.* Android Open Source Project. 2021.

[16] Paul Kocher et al. "Spectre Attacks: Exploiting Speculative Execution". In: *CoRR* abs/1801.01203 (2018). arXiv: 1801.01203. URL: http://arxiv.org/abs/1801.01203.

[17] Moritz Lipp et al. *Meltdown*. 2018. arXiv: 1801.01207 [cs.CR].

[18] PCAPDroid. *https://github.com/emanuele-f/PCAPdroid*.

[19] Qualcomm. *Qualcomm Adreno GPU Overview*. n.d.

[20] Lookout Mobile Security. *Lookout Pegasus Android Technical Analysis*. URL: https://info.lookout.com/rs/051-ESQ-475/images/lookout-pegasus-android-technical-analysis.pdf.

[21] Lookout Staff. *Pegasus for Android: The Other Side of the Story Emerges*. URL: https://www.lookout.com/threat-intelligence/article/pegasus-android.

[22] tshark. *https://www.wireshark.org/docs/wsug_html_chunked/AppToolstshark.html*.

[23] Quancheng Wang, Ming Tang, and Jianming Fu. "EavesDroid: Eavesdropping User Behaviors via OS Side Channels on Smartphones". In: *IEEE Internet of Things Journal* 11.3 (2024), pp. 3979–3993. DOI: 10.1109/JIOT.2023.3298992.

[24] Yingchen Wang et al. "GPU. zip: On the Side-Channel Implications of Hardware-Based Graphical Data Compression". In: *2024 IEEE Symposium on Security and Privacy (SP)*. 2024, pp. 84–84.

[25] Boyuan Yang et al. "Eavesdropping user credentials via GPU side channels on smartphones". In: *Proceedings of the 27th ACM International Conference on Architectural Support for Programming Languages and Operating Systems*. ASPLOS '22. Lausanne, Switzerland: Association for Computing Machinery, 2022, pp. 285–299. ISBN: 9781450392051. DOI: 10.1145/3503222.3507757. URL: https://doi.org/10.1145/3503222.3507757.

[26] Google Project Zero. *CVE-2019-2215: Android use-after-free in Binder*. 2019.