We know what you're doing! Application detection using thermal data

Philipp Miedl
Computer Engineering and Networks Laboratory, ETH Zurich,
Gloriastrasse 35, Zurich, Switzerland

Rehan Ahmed
Information Technology University of the Punjab,
Arfa Software Technology Park, Ferozpur Road, Lahore, Pakistan

Lothar Thiele
Computer Engineering and Networks Laboratory, ETH Zurich,
Gloriastrasse 35, Zurich, Switzerland

Abstract

Modern mobile and embedded devices have high computing power which allows them to be used for multiple purposes. Therefore, applications with low security restrictions may execute on the same device as applications handling highly sensitive information. In such a setup, a security risk occurs if it is possible that an application uses system characteristics to gather information about another application on the same device.

In this work, we present a method to leak sensitive runtime information by just using temperature sensor readings of a mobile device. We employ a Convolutional-Neural-Network, Long Short-Term Memory units and subsequent label sequence processing to identify the sequence of executed applications over time. To test our hypothesis we collect data from two state-of-the-art smartphones and real user usage patterns. We show an extensive evaluation using laboratory data, where we achieve labelling accuracies of up to 90% and negligible timing error. Based on our analysis we state that the thermal information can be used to compromise sensitive user data and increase the vulnerability of mobile devices. A study based on data collected outside of the laboratory opens up various future directions for research.

2012 ACM Subject Classification Security and privacy → Embedded systems security; Security and privacy → Hardware attacks and countermeasures; Security and privacy → Mobile platform security; Security and privacy → Virtualization and security; Hardware → Temperature monitoring

Keywords and Phrases Thermal Monitoring, Side Channel, Data Leak, Sequence Labelling

Digital Object Identifier 10.4230/LITES.7.1.2

Supplementary Material All data presented in this work, including the raw data collected from the smartphones, are available online under following links:

- `https://doi.org/10.3929/ethz-b-000418184`
- `https://doi.org/10.3929/ethz-b-000418183`

Experiment Orchestration Toolkit is available via the Experiment Orchestration Toolkit (ExOT)-project-website `exot.ethz.ch`, including the extension made in this work [28]. The specific tools used, as well as the code and configuration developed for this publications, can be found under the following links:

- `https://gitlab.ethz.ch/tec/public/exot/eengine/-/tree/pub_MAT20`
- `https://gitlab.ethz.ch/tec/public/exot/app_apk/-/tree/pub_MAT20`
- `https://gitlab.ethz.ch/tec/public/exot/app_lib/-/tree/pub_MAT20`
- `https://gitlab.ethz.ch/tec/public/exot/compilation/-/tree/pub_MAT20`

Before executing any experiments, please run the `RUNE.sh` script from the `eengine/data` directory, to reshape the data into the fitting form for this version of ExOT.

Acknowledgements Thanks to Xiaoxi He and Matthias Meyer for their valuable input and the discussion concerning the machine learning part of the work. Many thanks to Balz Maag for recording all the user inputs and the verification traces with both mobile phones, and Lukas Sigrist for providing access to the thermal-testbed. Thanks also to our former students Raphael, Max, Manuel and Athanasios for providing useful insights into thermal sequence data processing during their semester projects.

Received 2020-03-31 **Accepted** 2021-03-26 **Published** 2021-08-12

Editor Alan Burns and Steve Goddard

Special Issue Special Issue on Embedded System Security

1 Introduction

Due to their high computational power, mobile devices, such as smartphones and tablets, are often used for multiple purposes concurrently. Consequently, applications with different levels of security and privacy clearances reside on the same piece of equipment. For example, companies may allow their employees to use the same smartphone for business and private applications, or people with medical conditions may use their smartphone to monitor their health status. In both cases, applications performing highly-critical tasks operate beside benign applications, like games. To prevent security and privacy violations due to different applications on the same device, Operating Systems (OSs) often rely on the security paradigm of *application isolation and permission separation*. This paradigm defines that security and privacy are ensured if information transfer between applications is only possible under the oversight of the OS.

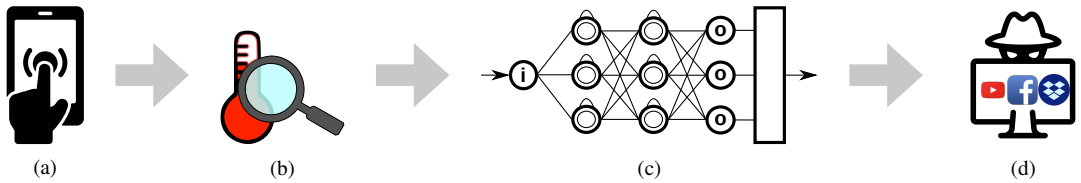
Previous work has shown that the security framework of application isolation and permission separation is susceptible to data leaks based on shared resources, such as caches [42]. While shared resources can be used to compromise the system, they can aim at increasing the efficiency of a system or providing monitored means of communication between applications. So even though shared resources might pose a security threat, we do not consider their presence in a system as a breach of the security paradigm of application isolation and permission separation.

An example for an implementation of application isolation and permission separation is Android application sandboxing[1]. Android sandboxing isolates applications and only allows applications to communicate through explicitly permitted channels. For example, applications can communicate using intents, which are subject to the scrutiny of the OS as all intents are routed through the Android system. While previous work has shown that intents can be misused to create data leaks [11], we argue that the principal design of intents does not violate the security paradigm of application isolation and permission separation.

In this work, we show how to bypass the security paradigm of application isolation and permission separation and compromise a system by providing a method that allows an adversary to determine which applications are executed on the device at specific time intervals.

The execution of applications influences the power consumption of the device and its temperature. Therefore, temperature readings may contain information on the executed application. Modern Systems-on-Chips (SoCs) typically have multiple thermal sensors to support smart power management. For example, Arm big.LITTLE SoCs often feature thermal sensor readings in both core clusters and several more for other components of the SoC. In general, these thermal measurements are exposed to the OS through an unrestricted software interface, which makes thermal data easily accessible. For instance, there are several applications on the Google Play

[1] `https://source.android.com/security/app-sandbox`



(a) (b) (c) (d)

Figure 1 (a) Different applications are executed on a mobile device depending on the user input. (b) Thermal information provided by the Operating System is collected by a third-party application. (c) Analysis of the thermal data to determine the application sequence. (d) The application sequence is used to create a usage profile or detect other applications, causing security and privacy violations.

Store that allow the thermal information of an Android device to be read, without the need for elevated privileges[2]. It has already been shown that these thermal readings may lead to security issues [6]. In particular, the thermal covert channel presented in by Bartolini et al. [1] shows the possibility of a data leak, as Ristenpart et al. [35] already stated that "Covert channels provide evidence that exploitable side channels may exist". However, as thermal information is vital to power management, it remains accessible without further permission requirements.

In this work, we present a side channel attack based on thermal sensor readings of a mobile device, as depicted in Figure 1. Different applications are executed on a mobile device based on the user input (a). These applications are not allowed to share any information without the supervision of the OS, due to the security paradigm of application isolation and permission separation. However, applications are allowed to read thermal information from the OS software interface. An adversary may deploy an application to read the thermal information (b) and analyses the thermal data (c). This allows the adversary to determine the execution sequence of other applications or detect the set of running applications (d). As this establishes an information transfer between intentionally isolated applications, such a thermal side channel violates the security and privacy restrictions imposed by the OS.

Contributions. Our main contributions in this work are:

1. We present a side channel attack that uses thermal data collected from mobile devices to determine patterns of application usage.
2. To the best of our knowledge, we are the first to apply machine learning techniques from time-series processing domain to determine an application execution sequence.
3. We present an extensive experimental evaluation of the thermal side channel based on real user interactions with the device, laboratory and real-world data.

2 Related work

Mobile devices are now present in almost all aspects of our daily life and have increased computing power. While usage information can be used to improve the functionality of mobile devices [5], this information can also be misused for malicious purposes [39]. This leads to security and privacy issues, which are well defined and broadly studied problems in current computing systems. Lampson [19] was the first to highlight the problem of *covert* and *side channels*, when he defined the *confinement problem*. The confinement problem states that an application is secure as long as it is ensured, that no information can be leaked to a third party. However, data leaks in form of

[2] e. g., Simple System Monitor (`https://play.google.com/store/apps/details?id=com.dp.sysmonitor.app`)

covert or side channels are almost omnipresent in modern computing systems, which makes their analysis even more important. In this work, we analyse a data leak, or side channel, based on the availability of temperature data.

2.1 Architectural data leaks

In the last years, many data leaks have been discovered and analysed, often based on architectural features of modern multicore systems. For example, the well known Spectre [18] and Meltdown [21] data leaks take advantage of the hyper-threading in Intel processors. Similarly, Rong et al. [36] showed how to compromise cloud systems by taking advantage of the so called Cloud Covert Channel based on Memory Deduplication (CCCMD). This channel has further been improved [24, 33] and specifically targeted at Intel SGX based systems [10]. Furthermore, a variety of other cache channels exploit the fact that multiple cores share the same cache [20]. However, also branch predictors [8] or random number generators [7] have been exploited as possible sources for data leaks in modern computing systems. Our work will not take advantage of architectural issues to establish a data leak, but exploit a data leak based on the thermal state of a mobile device.

2.2 Temperature related data leaks

Murdoch [31] showed that it is possible to identify servers in the tor network by analysing the temperature induced clock skew of the machines. The server was heated by causing a high load and was identified by concurrently analysing the timestamps of response packets of the server. This temperature induced timing side channel was further analysed and improved by Zander and Murdoch [44]. The authors minimized the clock jitter in the response packets and derived a channel capacity of approximately 20.5 bits per hour [43]. Furthermore, Ristenpart et al. [35] showed that this technique can also be applied to find other Virtual Machines (VMs) running on the same infrastructure. Other temperature related data leaks have been shown, which rely on the effects of the power management system of the devices. This includes fan speed [2], power consumption [26, 30] or operating frequency [27]. Similar to these work, we will also take advantage of the fact that modern computing devices have varying power needs and temperature profiles for different computing tasks.

Data leaks might also occur if chips are not operated in the right temperature range, as demonstrated by Hutter and Schmidt [15] on the RSA implementation on an AVR microcontroller. The authors showed that the hamming weight of the key can be leaked by correlating temperature, power and execution time, if the chip is operated beyond its specified temperature range. In this work, we will also use temperature as a medium to leak information, without the knowledge of the OS. However, we will not tamper with the environmental conditions and only rely on temperature measurements provided by the device.

2.3 Thermal data leaks and attacks

In addition to temperature related effects, temperature itself can serve as a medium to leak data. Islam et al. [17] presented a thermal side channel attack that allows an attacker to time power attacks on data centres more effectively. Thermal covert channels have also been extensively studied on FPGAs, where on-chip heat generators were used to transfer information out of the secure zone of the chip [22, 3, 16] or over time to the next scheduled application [40]. It has also been shown that general purpose CPUs are vulnerable to temperature based covert channels [23, 1]. Guri et al. [13] presented a method to establish such a thermal covert channel between air-gapped systems.

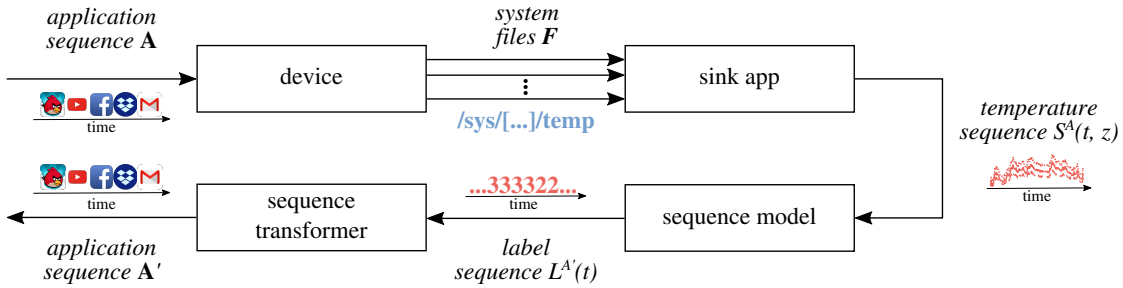


Figure 2 The information flow during a thermal side channel attack. A user executes a sequence of different applications $\mathbf{A}$ on the device. The sink application monitors the resulting heat generation from the device by reading the respective system files $\mathbf{F}$ for the different thermal zones $z \in \mathbf{Z}$. The sink application outputs the thermal sequence $S^A(t, z)$, which is fed to the sequence model. This model generates the label sequence $L^{A'}(t)$, holding one application label per time-step. $L^{A'}(t)$ is fed to the sequence transformer, which then outputs the inferred application sequence $\mathbf{A}'$.

2.4 Machine learning in security applications

Fuelled by the rapid advances in the machine learning domain, techniques from the machine learning domain are more frequently applied to device security and privacy relevant topics. Machine learning methods have been employed to build detectors for malicious applications. For example, the detectors analyse the Application Program Interface (API) calls and correlate it with the permission set of applications [32]. Buczak and Guven [4] gathered many other examples for machine learning applications in the domain of cyber security, classifying all these methods in three groups:

(i) misuse based approaches that identify known attacks by their signature,
(ii) anomaly based approaches which recognise behaviour which is not considered "normal", and
(iii) hybrid systems, which are a mix of misuse and anomaly based approaches.

All of these techniques rely on the availability of large amount of data for training.

In our work, we will also apply machine learning techniques in the security domain. While most of the existing works take advantage of machine learning techniques for defence mechanisms, we will use machine learning from the perspective of an adversary. Furthermore, we will show how to generate a large amount of data for training, without having to deploy a complex measurement system for a long period of time.

3 Threat model

First, we describe the attack scenario to define the threat model of the thermal side channel attack. Second, we present the attack concept to outline the techniques used to mount a thermal side channel attack.

3.1 Attack scenario

We base our threat model on the scenario presented in Figure 1 and the information flow in Figure 2. A user runs a sequence of applications $\mathbf{A}$ on a mobile device. This sequence consists of an arbitrary sequential order and duration of application executions, depending on the user needs. Due to the difference in computational effort and the components utilised by different applications, the device generates different heat patterns in time and space. These heat patterns can be determined by observing the different thermal zones of a device. A thermal zone defines

sensor readings for a specific part of a device, for example, a processor core. An adversary infiltrates the mobile device, for example, by disguising a thermal monitoring *sink* background service as part of a benign game. Therefore, the adversary is able to monitor the temperature sensors of the mobile device.

The sink in Figure 2 collects a temperature sequence $S^A(t, z)$, which is composed of the thermal readings of all observed thermal zones $z \in \mathbf{Z}$. The adversary analyses the temperature sequence $S^A(t, z)$ using a sequence model. As a result of the analysis, the adversary obtains an application label sequence $L^{A'}(t)$ with one label per time-step. In the final analysis step, the per-time-step label sequence $L^{A'}(t)$ is transformed into a condensed application label sequence $\mathbf{A'}$. This transformation is necessary to eliminate duplicate labels and artefacts in the per-time-step label sequence $L^{A'}(t)$. Finally, the output application sequence $\mathbf{A'}$ allows an adversary to derive further insights into the user behaviour with different security and privacy implications. These implications depend on whether the adversary does the analysis *offline* or *online*.

Offline Scenario. In this scenario, the adversary only deploys the sink application on the attacked device. The sequence model, as well as the sequence transformer, are implemented on a dedicated analysis device. Therefore, the sink application transfers the thermal data to the analysis device for application sequence inference. This way, the attacker can determine which applications were executed at what time and create a detailed user profile containing sensitive information. For example, the usage of medical applications would allow inferences on the medical condition of the user, or location-based applications, e. g., a regional tourism application, allow inferences about the location and activities of the user. Such a data leak would present a major privacy violation, as the profiling of the user behaviour could be performed without the user's knowledge or consent.

Online Scenario. In the online attack scenario, the attacker performs the application sequence inference on the attacked device in real-time. This means, in addition to the sink applications, the sequence model and the sequence transformer also have to be deployed on the attacked device. The attacker may then use the real-time information to time a targeted attack on a specific application in the secure domain. Such an attack is dangerous when considering a company that enforces the *bring-your-own-device* policy. According to this policy, employees will use their mobile devices for private and business applications. Therefore, the phone features two application domains, i. e., business and private, which are separated by virtualisation. An example of a virtualisation environment is "Android for work". Such a system is vulnerable if an attacker can retrieve information about applications in the secure domain. This can be achieved if an attacker mounts an online thermal side channel attack in the less secure domain to gain runtime information from the secure domain.

3.2 Concept of a thermal side channel attack

To mount the thermal side channel attack, the sink application, the sequence model and the sequence transformer need to be implemented. We base the sink application design on ExOT, presented by Miedl et al. [29], which allows us to sample the thermal zones of a device in a timed fashion and without the need for elevated privileges.

The thermal side channels establish a highly complex transformation from device usage patterns to observable temperature changes in the various thermal zones $z \in \mathbf{Z}$. In addition, the usage patterns of applications differ vastly, as do the interactions of applications with their users. Masti et al. [23] already stated that correlation based methods show very limited capabilities as thermal features identifying an application are very subtle. Therefore, we use techniques from the machine learning domains of sequence-to-sequence labelling and time-series modelling to implement the

sequence model. We build our sequence model using a Convolutional-Neural-Network (CNN) and a Recurrent Neural Network (RNN), which we describe in detail in section 5. The sequence transformer is based on classical filtering algorithms and rules of condensing labels, as outlined in section 6.

It is well known that CNNs and RNNs require a tremendous amount of labelled training data to perform well. In our case, suitable data should be available to represent user interactions and to represent thermal traces from different applications on different devices and in different thermal environments. However, there is no appropriate dataset available, nor is it feasible to deploy a long-term measurement setup to gather sufficiently diverse data (user interactions, applications, devices, thermal environments, interferences) and label them correctly. Therefore, we must define a data augmentation scheme that allows us to generate a highly representative and diverse data set.

4 Data augmentation

The overall process to generate a large set of diverse thermal sequences is depicted in Figure 3. It contains four distinct components that will be detailed in the subsequent sections. Input to the data augmentation scheme are

(i) the device for which the thermal sequence needs to be generated,

(ii) the set of applications that will potentially run and corresponding user inputs, and

(iii) a dataset configuration, which contains information to configure the whole generation scheme.

The device characterisation uses measurements in order to model the thermal behaviour of the device itself. The outcomes are temperature coefficients, namely, the internal thermal, ambient heating and ambient cooling coefficients, which are used to concatenate the thermal profiles of applications and augment the data by modelling changes in the ambient temperature of the device. Note that these coefficients are determined for each temperature sensor (or zone) z individually.

The purpose of the application characterisation is to record the temperature trace of a running application. This is characteristic for the application and can be used by an adversary application to spy on it. Again, the traces are collected for each temperature zone individually.

The sequence parameter generation determines (random) sequences of applications whose corresponding temperature sequences are generated by the thermal sequence generation. In order to increase the diversity of training data, it also (randomly) generates traces of ambient temperature offsets in order to model that the device is exposed to dynamically changing external temperatures.

The thermal sequence generation combines all of this information to generate a temperature sequence and its associated label sequence.

We base our thermal modelling on *Newton's law of cooling*. It states that the rate of heat loss of a body is directly proportional to the difference in the temperatures between the body and its surroundings. Therefore, it is expected that the system will experience exponential decay in the temperature difference of body and surroundings as a function of time. Note that Newton's law does not take heat transfer between individual architectural elements into account. However, as the experimental results show, this approximation is sufficiently accurate for our purpose.

The basic form of Newton's law is

$$T(t_2) = T^{\text{idle}} + (T(t_1) - T^{\text{idle}}) \cdot e^{-\beta(t_2 - t_1)} \tag{1}$$

where β denotes the thermal coefficient, $T(t)$ denotes the temperature at time t, we have $t_2 \geq t_1$, and T^{idle} denotes the steady-state temperature. For convenience, we introduce the temperature

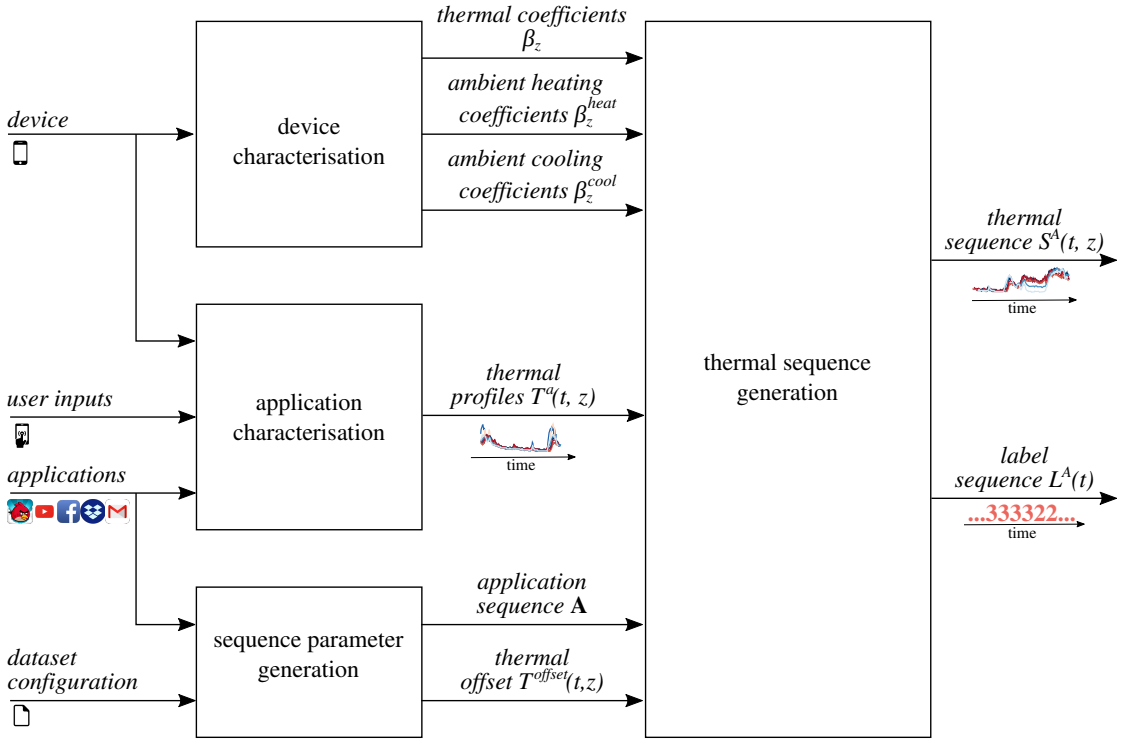


Figure 3 Overall data augmentation scheme structure. We generate a thermal sequence and its labels based on (i) the particular device and its measured thermal characterisation, (ii) the set of identifiable applications and their thermal characterisations, and (iii) the highly-configurable data set configuration. Using diverse configurations, we generate representative training data.

function $C(\cdot)$ with

$$\Delta T(t_0 + t) = C(\Delta T(t_0), \beta, t) = \Delta T(t_0) \cdot e^{-\beta \cdot t} \tag{2}$$

It returns the temperature difference to the steady-state temperature $\Delta T(t_0 + \Delta t) = T(t_0 + \Delta t) - T^{\text{idle}}$ after time t and uses the initial temperature difference $\Delta T(t_0) = T(t_0) - T^{\text{idle}}$, the thermal coefficient β, and the time difference t. In order to be able to consider different heat transfer mechanisms that mediate between heat losses and temperature differences, we can use different constant thermal coefficients in the temperature function $C(\cdot)$.

4.1 Device characterisation

One basic component of the data augmentation scheme is the characterisation of each device in terms of its thermal coefficients. This model will be used later on to combine the temperature profiles of two applications that run in sequence.

In order to consider different heat transfer mechanisms, we will characterise each thermal zone $z \in \mathbf{Z}$ of a given device by three thermal coefficients:

1. the internal temperature coefficient β_z,
2. the ambient heating coefficient β_z^{heat}, and
3. the ambient cooling coefficient β_z^{cool}.

The internal temperature coefficient β_z describes how long the heating effect from a previously-executed application continues. To approximate this coefficient, we conduct measurements on the respective device. The device is placed in an environment with the controlled ambient temperature

and kept idle before the start of the measurement. Next, a benchmark application is executed for some randomly chosen time, which increases the temperatures of the zones by some value. As soon as the application is stopped, a temperature trace is recorded. The measurement is repeated for different run-times of the benchmark. From these traces, we derive the internal temperature coefficient β_z for each thermal zone z using regression analysis of the temperature traces, i. e., fitting the temperature function $C(\cdot)$ in Equation (2) to the temperature measurements using non-linear least squares.

The two ambient coefficients for heating and cooling, β_z^{heat} and β_z^{cool}, respectively, model the influence of increasing or decreasing ambient temperatures on a specific thermal zone. Both coefficients are determined by first moving the idle device to an environment with a different ambient temperature, then moving it back to the initial environment and taking the corresponding temperature trace. By isolating the time intervals when the device adapts to the new ambient temperature, similar to β_z, we use regression analysis to determine β_z^{heat} and β_z^{cool}.

4.2 Application characterisation

We characterise an application a running on a chosen device by determining the *thermal profile* $T^a(\Delta t, z)$, which describes the thermal behaviour caused by the execution of the application a. Here, Δt denotes the time since the application start and z denotes the observed thermal zone of the device.

To derive the thermal profile of an application, we record a thermal trace $T^a(t, z)$ during the execution of the application a, i. e., $0 \leq t \leq t_{\text{exec}}$. We do this measurement in the same environment as before for the internal thermal coefficients β_z. This way, we can ensure that the device has settled before starting the measurement and minimise the chance for interference from external factors.

For every application, we perform multiple measurements to derive multiple thermal profiles. This allows us to acquire a more diverse set of profiles and compensate for thermal variations caused by the measurement setup. In addition, we also record multiple user interaction patterns for each application, encapsulating typical application use cases. The user input for the different use cases of the applications is recorded and replayed, to ensure reproducibility of the raw data.

4.3 Sequence parameter generation

The component for sequence parameter generation provides a sequence of applications and a trace of changing ambient temperature offsets. Depending on these inputs, the corresponding thermal sequence is generated. By changing the sequence of applications and the ambient temperature offset of the device in an appropriate way, a large set of diverse thermal sequences and their associated labels can be generated. The strategy by which the sequence parameters are generated is described in the dataset configuration. In the following, we describe the functionality of the sequence parameter generation.

Based on the available applications a and the dataset configuration, we generate the application sequence as an ordered list of tuples $\mathbf{A}$. The i^{th} tuple of $\mathbf{A}$ is defined as $< a_i, t_i^{\text{start}}, t_i^{\text{end}} >$, where a_i defines the i^{th} application which is started at time t_i^{start} and closed at time t_i^{end}. We do not consider the concurrent execution of two or more applications. Therefore, the execution intervals of individual applications are assumed to be disjoint and consecutive: $t_{i+1}^{\text{start}} = t_i^{\text{end}}$ and $t_i^{\text{end}} > t_i^{\text{start}}$. This assumption is realistic, considering the typical usage of a mobile device where a single foreground application is executed at a time. Nevertheless, one should also mention that there are also other usage patterns where music can play in the background, or two applications can be used side-by side in split-window mode.

Depending on the information in the dataset configuration, we generate the application sequence **A** either

(A) randomly,

(B) systematically to increase the number of different thermal profile sequences in the data set, or

(C) such that thermal profiles are randomly chosen from two sets of profiles, alternating.

Method (B) ensures that the maximum number of different thermal profiles appear in the data set. Method (C) is used, for example, if the thermal profiles can be split into two groups, i. e., thermal profiles of known and unknown applications.

The thermal offset $T^{\text{offset}}(t, z)$ simulates different environmental temperatures for different locations of the mobile device, e. g., indoors or outdoors. To generate a relative offset trace $T^{\text{offset}}(t, z)$ for each temperature z, the sequence parameter generation first randomly generates an initial thermal ambient thermal offset $\Delta T_0^{\text{ambient}}$ and an ambient thermal offset sequence $\mathbf{\Delta T^{ambient}}$ with n tuples. The i^{th} tuple of the sequence is defined as $< \Delta T_i^{\text{ambient}}, t_i^{\text{enter}} >$ and specifies that the device enters an environment with the ambient thermal offset of T_i^{ambient} at time t_i^{enter}. The ambient thermal offset is relative to the ambient temperature in the measurement environment used for the application characterisation (see above). Based on this ambient thermal offset sequence, we derive $T^{\text{offset}}(t, z)$ using Newton's law as follows:

$$T^{\text{offset}}(t,z) = \begin{cases} \mathcal{U}(-1,1) + \Delta T_0^{\text{ambient}} & \forall\ 0 \le t < t_1^{\text{enter}} \\ \mathcal{U}(-1,1) + \Delta T_i^{\text{ambient}} + C\left(T^{\text{offset}}(t_i^{\text{enter}}, z) - T_i^{\text{ambient}}, \beta_z^{\text{heat}}, t - t_i^{\text{enter}}\right) & \cdots \\ \qquad \forall\ 1 \le i \le n\ \wedge\ t_i^{\text{enter}} < t \le t_{i+1}^{\text{enter}}\ \wedge\ T_{i-1}^{\text{ambient}} \le T_i^{\text{ambient}} & \\ \mathcal{U}(-1,1) + \Delta T_i^{\text{ambient}} + C\left(T^{\text{offset}}(t_i^{\text{enter}}, z) - T_i^{\text{ambient}}, \beta_z^{\text{cool}}, t - t_i^{\text{enter}}\right) & \cdots \\ \qquad \forall\ 1 \le i \le n\ \wedge\ t_i^{\text{enter}} < t \le t_{i+1}^{\text{enter}}\ \wedge\ T_{i-1}^{\text{ambient}} > T_i^{\text{ambient}} & \end{cases} \tag{3}$$

where $t_{n+1}^{\text{enter}} = \infty$. Note that when the ambient temperature increases, we use β_z^{heat} as the thermal parameter for the thermal behaviour model of the zone, while for decreasing ambient temperatures, we employ β_z^{cool}. $T^{\text{offset}}(t, z)$ not only contains the ambient offset of the thermal trace but also adds random thermal noise $\mathcal{U}(-1, 1)$, where -1 and 1 define the maximum amplitude of the thermal noise in °C. This is necessary, as traces generated from laboratory data is less noisy than real-world recordings.

4.4 Thermal sequence generation

Now that we have chosen the sequence parameters and have characterised the device, as well as the applications, we are in the position to generate a corresponding *thermal sequence*.

The first challenge is to generate application sequences based on the provided temperature profiles. The simple concatenation of these profiles following the provided application sequence **A** is not possible for three reasons: First, the final temperature of a temperature profile does not typically match the initial temperature of the subsequent application. Second, we cannot expect that an application runs from start to end. Rather, it undergoes a halting or closing phase when a context switch takes place. Finally, we must consider the changing ambient temperature offset, as well as the initial temperature offset of the thermal sequence. We will now go through these challenges one-by-one.

Before we can concatenate the thermal profiles according to the generated application sequence **A**, we have to select and crop thermal profiles to the desired length. However, we must include the thermal information of closing or halting the application in the cropped thermal profile. To this end, we empirically evaluate the time interval at the end of a thermal profile necessary to close an application. For example, a thermal profile has the length of 20 s, and we have evaluated

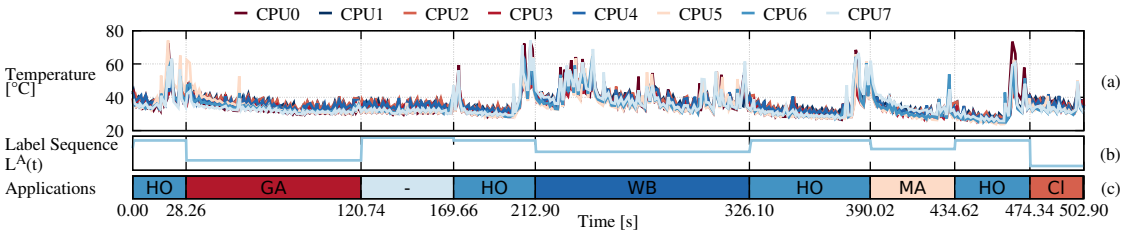


Figure 4 Thermal sequence build from different thermal profiles collected from a Sony Xperia Z5, see Equation (6). (a) illustrates the thermal traces, (b) the label sequence $L^{\mathrm{A}}(t)$ and (c) the application trace $\mathbf{A}$. The application labels correspond to Table 1. Our data augmentation removes temperature discontinuities at application pre-emption points without distorting the thermal profiles.

that the last 4 s are used to close the application. If we now want to crop the thermal profile to a length of 12 s, we crop it to 8 s and then append the final 4 s. To ensure that there are no temperature discontinuities in the cropped thermal profile, we employ the temperature function $C(\cdot)$ as follows:

$$\overline{T}^a(t,z) = \begin{cases} T^a(t,z) & \forall\, t \leq t_{crop} - t_{close} \\ T^a(t + t_{exec} - t_{crop}, z) + & \cdots \\ \quad C\left(T^a(t_{crop} - t_{close}, z) - T^a(t_{exec} - t_{close}), \beta_z, t + t_{close} - t_{crop}\right) & \\ \quad \forall\, t_{crop} - t_{close} < t \leq t_{crop} & \end{cases} \tag{4}$$

Here, t_{crop} is the cropped length the thermal profile, i. e., the length as requested from the application sequence, t_{close} is the time interval needed for closing an application, and t_{exec} the total length of the thermal profile. We note that $t_{close} < t_{crop} < t_{exec}$.

Now, we will determine the thermal sequence $S^A(t,z)$ of an application sequence $\mathbf{A}$ with tuples $< a_i, t_i^{\text{start}}, t_i^{\text{end}} >$ for $1 \leq i \leq n_A$, while

(i) considering that there are no discontinues in the sequence when switching from one application to the next,

(ii) taking into account that the ambient temperature is changing, and

(iii) setting the initial temperature of the thermal sequence.

We obtain the thermal sequence $S^A(t,z)$ for each temperature zone z as follows:

$$S^A(t,z) = \begin{cases} T^{\text{offset}}(t,z) + \overline{T}_1^a(t,z) & \forall\, 0 = t_1^{\text{start}} \leq t \leq t_1^{\text{end}} \\ T^{\text{offset}}(t,z) + \overline{T}_i^a(t,z) + C\left(S^A(t_{i-1}^{\text{end}}, z) - \overline{T}_i^a(0,z), \beta_z, t - t_i^{\text{start}}\right) & \\ \quad \forall\, 2 \leq i \leq n_A \;\wedge\; t_i^{\text{start}} < t \leq t_i^{\text{end}} & \end{cases} \tag{5}$$

Here, we have n_A applications with indices $1 \leq i \leq n_A$, where the first application starts at time $t_1^{\text{start}} = 0$. Note that we use the cropped temperature profiles, as determined in Equation (4). We employ the temperature function $C(\cdot)$ to offset every thermal profile in accordance with our temperature model so that there are no discontinuities in the thermal trace at the concatenation points. As a final step, we derive the label sequence $L^A(t)$ corresponding to the thermal sequence $S^A(t,z)$. The label sequence $L^A(t)$ is the numerical representation of the application label for each time step.

Figure 4 illustrates the thermal sequence generated from the example application sequence $\mathbf{A}$ defined as

$$\begin{aligned} \mathbf{A} = (\ &< \text{HO}, 0.00\,\text{s}, 1.13\,\text{s} >, < \text{GA}, 1.13\,\text{s}, 4.83\,\text{s} >, < \text{-}, 4.83\,\text{s}, 6.79\,\text{s} >, \\ &< \text{HO}, 6.79\,\text{s}, 8.52\,\text{s} >, < \text{WB}, 8.52\,\text{s}, 13.04\,\text{s} >, < \text{HO}, 13.04\,\text{s}, 15.60\,\text{s} >, \\ &< \text{MA}, 15.50\,\text{s}, 17.38\,\text{s} >, < \text{HO}, 17.38\,\text{s}, 18.97\,\text{s} >, < \text{CL}, 18.97\,\text{s}, 20.22\,\text{s} >) \end{aligned} \tag{6}$$

The application labels are outlined in Table 1, and the thermal profiles were collected from a Sony Xperia Z5 smartphone and the settings outlined in section 7. The example illustrates that our data augmentation scheme is able to generate traces that look realistic and do not show any discontinuities.

5 The sequence model

In this section, we show how we can apply well-known methods from time-series and sequence-to-sequence modelling to mount a thermal side channel attack. In particular, we describe an implementation of the sequence model, as shown in Figure 2. Its purpose is to transform the received thermal sequences from all the thermal zones $S^A(t,z)$ into a sequence of labels $L^{A'}(t)$, i. e., the sequence of presumably running applications. We describe the chosen basic neural network architecture, as well as the training setup.

5.1 Neural Network architecture

The thermal side channel is characterised by complex, largely unknown and non-deterministic properties. First, the running applications can only be identified through their time-dependent usage pattern of the various components of the device such as its CPUs, GPU, memory and dedicated processing components. This usage pattern is unknown and non-deterministic, as it depends on the interaction of the user with the application, as well as data input. Second, the usage pattern leads to distributed power consumption that is converted to temperature changes and heat diffusion through an unknown thermal model of the device. Last but not least, interferences from changing ambient temperature, air flow, running software services, as well as measurement noise change the temperature pattern received by the sink application.

The transformation of a running application to a corresponding thermal sequence involves long-term and state-dependent behaviour. For example, an application can be identified by a sequence of usage patterns of the various components of the device and, therefore, memorising and identifying this sequence of usage patterns is an essential prerequisite for the sequence mode. The transfer from usage patterns to the thermal response at the thermal zones of the device also involves long-term state dependencies. In this case, the thermal energy that is diffusing. As a result, an effective sequence model needs to be able to flexibly represent state-dependent behaviour internally, i. e., it should have an internal state.

Due to this complexity of the input data, we choose a Neural Network (NN) based sequence model, which is able to learn the relation between running applications and the received thermal sequence. We compose this NN based sequence model using a feed-forward Convolutional-Neural-Network (CNN) for feature extraction and a Long Short-Term Memory (LSTM) based Recurrent Neural Network (RNN) to obtain the temporal relation between the thermal features. We give a brief overview of these two NN types in the following two sections.

5.1.1 The Convolutional Neural Network

A Convolutional-Neural-Network (CNN) is one of the techniques used in time-series processing, for example, seismic data [25]. As stated by Goodfellow et al. [9, Chapter 9], a CNN implements the convolution operation to extract feature maps from the input using kernels, defined as

$$y(t) = (x * k)(t) = \sum_{i=-\infty}^{\infty} (x(i) \cdot k(t-i)) \tag{7}$$

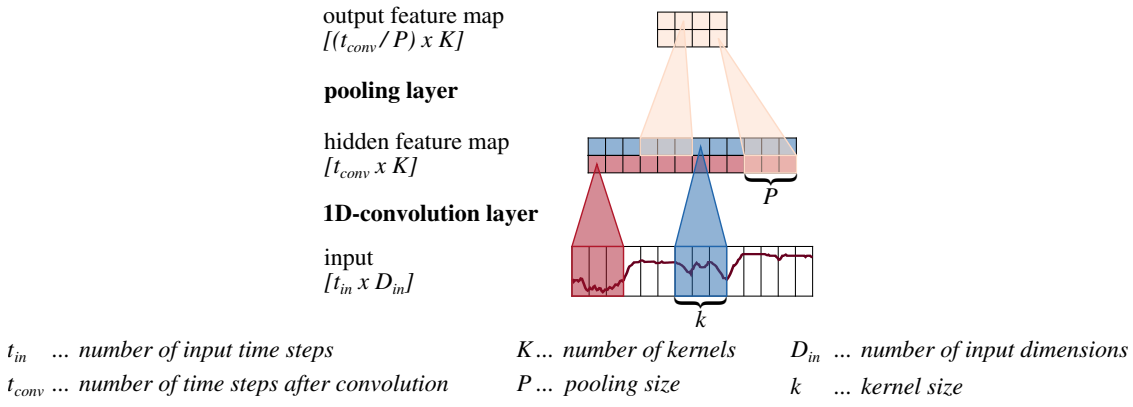


Figure 5 A simplified example of a one-dimensional Convolutional-Neural-Network (CNN). Note that the number of time steps after the 1D-convolution t_{conv} depends on the kernel size k and the data padding used for the convolution.

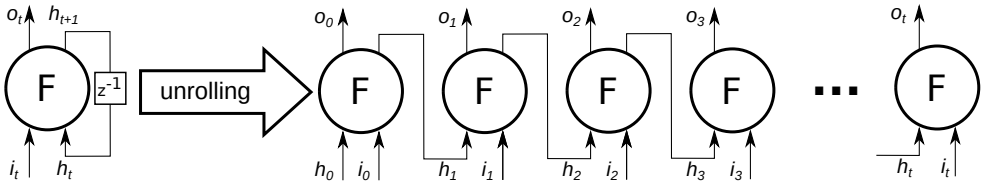


Figure 6 A Recurrent Neural Network (RNN) cell, in simple representation (left), and unrolled over time. i_t represents the input, o_t the output and h_t the internal state at time t.

where $x(t)$ is the input and $k(t)$ is the kernel. In our model, we employ one-dimensional feed-forward convolutional layers to transform the input to feature maps. Furthermore, we use pooling layers to reduce the size of the convolutional layer output. Roughly speaking, a CNN can efficiently transform our time-series data into a suitable format for further processing, while reducing the data size and thus reducing the complexity of following computation steps.

Figure 5 illustrates an example of such a CNN. The input data has a length of t_{in} time steps and D_{in} dimensions. Different dimensions can be, for example, different sensors that are read. The 1D-convolutional layer processes the input data to obtain the so-called hidden feature map, which has a size of $[t_{conv} \times K]$. Here, t_{conv} defines the number of time steps in the data after the convolution operation, which depend on the size of the kernel k and the padding method that is used. K defines the number of different kernels, or filters, used by the 1D-convolutional layer, illustrated by the different colours in Figure 5. The hidden feature map is downsampled by the pooling layer using a fixed pooling size P and a pre-defined function, for example, max. The pooling layer outputs the final feature map of size $[(t_{conv}/P) \times K]$.

5.1.2 The Long Short-Term Memory based Recurrent Neural Network

In contrast to simple feed-forward neural networks, RNNs have internal feedback loops that allow them to capture, represent and use temporal information in the input sequences. Figure 6 illustrates an RNN consisting of a single cell, feeding the internal state of the current time-step to the next time-step. For training, the RNN is unrolled over time (see Figure 6), to perform a *back-propagation* through time for computing gradients. In other words, after unrolling, the input vector dimensions of the RNN are extended by the time dimension. For example, a RNN that takes an input vector with N_F feature dimensions and is unrolled N_T time-steps, will take an $N_T \times N_F$ input matrix during training.

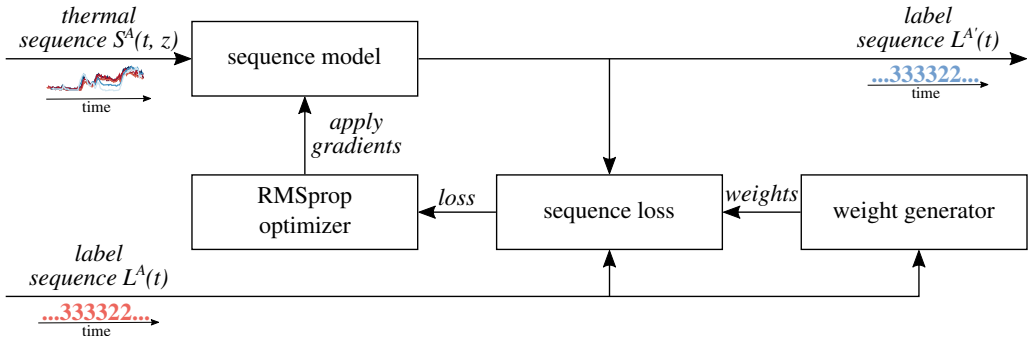


Figure 7 Setup during the training phase of the sequence model.

The main issue of simple RNN cells is the *vanishing-gradient-problem*, which does not allow them to capture long-term temporal relations in the data [12, Section 3.2]. This problem arises because, as the weights are shared for all time steps, input values are multiplied with the same weights, which leads to exponential decay of the sensitivity of the nodes towards the input data. To compensate the vanishing-gradient-problem, Hochreiter and Schmidhuber [14] designed Long Short-Term Memory (LSTM) cells with an input, a forget and an output gate. These gates allow LSTMs to control the information flow over time (long-term memory) better and, therefore, compensate for the vanishing-gradient-problem [12, Chapter 4].

LSTMs are often implemented as bi-directional networks, which capture temporal relationship in the data in both directions. Simply speaking, bi-directional networks consist of two sub-networks, one of which traverses the data from past to future and the other from future to past [12, Section 3.2.3]. Bi-directional networks often perform better, but they are more complex due to the increased size of the network. In addition, bi-directional networks cannot be used on-line as they require information from the future to process a label output. Therefore, we will use simple uni-directional LSTM layers in our sequence model.

5.2 Model structure and training setup

For our experiments, we use a network consisting of

(i) one convolutional layer with 64 filters and a kernel size of 25 (1 s),
(ii) one max-pooling layer,
(iii) 4 LSTM-layers with 128 units each, and
(iv) a dense layer with as many units as labels in the experiment scenario.

The CNN reduces the amount of data fed to the LSTM layers and extracts the most important thermal features. Using the LSTM layers, we derive and memorise timing-related information from the internal thermal feature stream. Lastly, the dense layer converts the LSTM layer output to a one-hot-encoded output label vector.

Figure 7 illustrates the setup for training of the sequence model. The input for the training are the thermal sequence $S^{\mathrm{A}}(t, z)$ and the corresponding label sequence $L^{\mathrm{A}}(t)$, which we generate as described in section 4.

During training, we use the *sequence loss* from the TensorFlow 2.0 Addons `seq2seq` package[3]. This loss function allows us to weight the individual samples of the trace for two purposes:

[3] `https://www.tensorflow.org/addons/api_docs/python/tfa/seq2seq/sequence_loss`

(i) if the input trace length is shorter than the actual input length of the model, we use zero weights to indicate which samples should be ignored, and

(ii) the weights allow us to compensate if some labels appear disproportionately often in the training data to ensure the training puts the same emphasis on all labels (example-weighted training).

The *weight generator* generates the weights depending on how often a label occurs in a batch. For example, in a batch label with two labels A and B, A occurs twice as often as B. In this case, the weights are 0.5 for label A and 1 for label B. The determined loss is then fed to the *RMSprop optimiser*, an adaptive learning rate optimiser which has proven to work well for many applications [41]. We choose the RMSprop optimiser because it is widely used and has been empirically evaluated to outperform simple stochastic gradient decent in terms of training time[4]. However, we acknowledge criticism towards stochastic optimisation techniques that has been raised in the past [34], but leave an evaluation of different optimisers for future work.

To avoid over-fitting, we use dropout layers and early stopping during training. The early stopping is triggered whenever the loss on the test set does not decrease, and the per-time-step accuracy does not increase after 20 epochs of training.

6 Sequence transformation and performance metrics

Following the information flow, as depicted in Figure 2, the sequence model transforms the thermal sequence into a label sequence. Due to the limited view of the sequence model on the relation between the application sequence and the resulting temperature sequence, we can observe artefacts when switching between applications. Finally, in order to evaluate the difference between the initial application sequence and the predicted application sequence, appropriate metrics need to be defined.

6.1 Sequence transformation

After training the sequence model, we feed an example thermal sequence to obtain the label trace, as illustrated in Figure 8. The example is generated using thermal profiles collected from a Sony Xperia Z5 smartphone, using the experimental setup outlined in section 7. In this section, we highlight issues in the label trace $L^{A'}(t)$ as produced by the sequence model and provide an approach to address them. Figure 8 illustrates a thermal sequence using the applications outlined in Table 1 and thermal profiles from a Sony Xperia Z5 (see section 7 and section 8). (a) shows the thermal sequence $S^{A}(t, z)$ for one of the zones, (b) the output label trace $L^{A'}(t)$ from the time-sequence model, (c) the output application sequence $\mathbf{A'}$, and (d) the ground truth $\mathbf{A}$.

The literature offers a variety of different approaches for a transformation from a label per time unit to a label interval. For example, combining the RNN with a Hidden Markov Model (HMM) or using the so-called Connectionist Temporal Classification (CTC) algorithm [12]. CTC based models provide an output distribution over all possible application sequences for a given input. One can use this distribution either to infer a likely application sequence or to assess the probability of a given one. However, as such advanced approaches require a considerable amount of training and computing overhead, we resort to the following simpler two-step approach: Use a filter to avoid the jitter label and a label condensing rule to convert $L^{A'}(t)$ into $\mathbf{A'}$. We consider this simple approach sufficient for our application, as we do not need to differentiate whether

[4] https://towardsdatascience.com/understanding-rmsprop-faster-neural-network-learning-62e116fcf29a

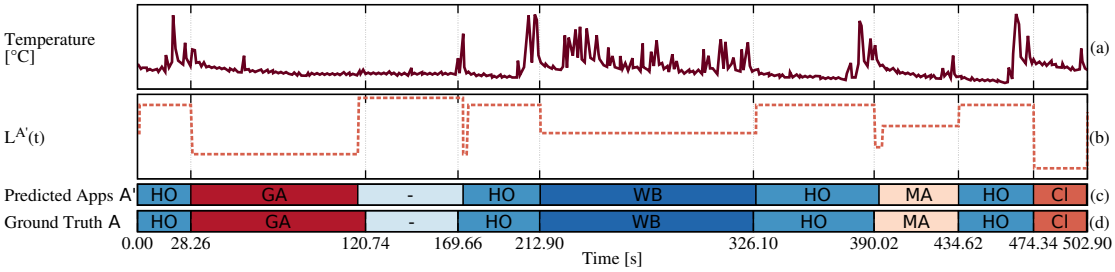


Figure 8 A thermal sequence processing example taken from the evaluation in section 8 using thermal profiles collected from a Sony Xperia Z5. shows (a) the thermal sequence $S^A(t, z)$ for one of the zones, (b) the labelling sequence $L^{A'}(t)$, (c) the predicted application sequence $\mathbf{A}'$ and (d) the actual application sequence $\mathbf{A}$. The application Labels correspond to the labels outlined in Table 1. The plot shows that the sequence model is capable of predicting correct labels with only a small amount of timing inaccuracy. Yet, labelling artefacts at the application pre-emption points at 169.66 s and 390.02 s occur. However, the sequence transformation using a majority voting filter and label condensing is able to compensate for such labelling artefacts.

an application has been executed multiple times in a row or once for a longer period of time. Nevertheless, the parameter of the filter should match the minimal time period of an application to be recognized.

In Figure 8, the output label trace $L^{A'}(t)$ at the pre-emption points at 169.66 s and 390.02 s jumps between multiple labels. Therefore, there is a need for an additional filter to eliminate such a label jitter. We apply a simple sliding window with majority voting replacement with a window length of 25 s as the minimal length of an application execution is larger. This window size is sufficiently large to compensate the label jitter and other labelling artefacts, yet small enough to allow for a sufficient temporal accuracy of the labelling.

After determining the filtered per-time-step labelling sequence $L^{A'}(t)$, we have to apply a final transformation to derive the predicted application sequence $\mathbf{A}'$. All consecutive equal labels are combined to a single label. By tracing the start and end time of the condensed labels, we can determine when an application is executed. The example from Figure 8 shows that the network is capable of determining the correct application sequence and that the timing is reasonably accurate. The latency of the labels at the application pre-emption points can be considered normal as Graves [12] already stated that uni-directional LSTM models often set the labels with some delay. However, the example also shows that if the thermal trace does not contain sufficient thermal features, misclassifications might happen. As the gaming application (GA) seems to be idle at the pre-emption point at 120.74 s, the sequence model sets the unknown label ("-") too early.

6.2 Performance metrics

In order to evaluate the performance of the whole approach as outlined in Figure 2, we need to determine different metrics. First, we derive the per-timestep accuracy of the label sequences $L^A(t)$ and $L^{A'}(t)$, which are of equal length. The per-timestep accuracy is the number of equal elements in $L^A(t)$ and $L^{A'}(t)$, divided by the length of $L^A(t)$.

To get a more detailed understanding of the performance of our approach, we also evaluate the difference between the initial application sequence $\mathbf{A}$ and the predicted one $\mathbf{A}'$. The associated challenges are due to the following characteristics:

(i) two application sequences $\mathbf{A}$ and $\mathbf{A}'$ might have different lengths,
(ii) element-wise comparison of two sequences may lead to misleading results, and
(iii) we are interested in the temporal correctness of the predicted application sequence in addition.

For example, the element-wise comparison of the correct sequence "ABDABABAB" and predicted sequence "ABDBABAB" yields a relative error of 5/8 when taking only the shorter application sequence as a reference. Besides the problem of different sequence lengths, we can also observe that there is just one difference in the two sequences, namely the missing "A" after "D". A metric that can handle sequences of different lengths is the relative *Levenshtein distance*, also known as relative edit distance. The relative Levenshtein distance is the minimal number of modifications that have to be applied to a sequence to be equal to another one, divided by the length of the correct sequence. Possible modifications are insert, delete and replace. For our example sequences, the relative Levenshtein distance is 1/8 as we just need to insert the application "A" after "D". This result shows that the two sequences are rather similar in terms of the relative Levenshtein distance, which is closer to our intuition.

The second metric we use for the final evaluation is the average timing error or temporal label placement error. A naive approach would measure the time error of the label placement by calculating the *Euclidean distance* between the predicted application pre-emption points and the actual pre-emption points. However, the predicted application sequence can contain a different number of application pre-emptions than the real application sequence. Therefore, we combine the Euclidean distance with the Dynamic Time Warping (DTW) algorithm [37]. DTW will calculate the Euclidean distance between all pre-emption points reported in the predicted application sequence, with the most similar pre-emption point in the true application sequence, and report the sum of all distances calculated for the sequences. For example, let us assume the network reports three application pre-emptions at 5 s, 8 s and 11 s, while the true sequence only contains two pre-emption points at 5 s and 9 s. Using DTW, the reported Euclidean distance will, therefore, be 0 s $+1$ s $+2$ s $= 3$ s. To normalise the time error metric, we divide the value reported by the DTW algorithm by the number of actual pre-emptions in the true sequence. This would result in an average time error of 1.5 s in the example. If a predicted application sequence only contains one application and, therefore, no pre-emption point, the average timing error is NaN.

7 Target Setup

The final performance evaluation of the thermal side channel attack is based on data from real smartphones. We chose two different smartphones from two different vendors for our evaluation:

- A Samsung Galaxy S5 SM-900H based on a Samsung Exynos 5422 SoC, with Android 5.0 and 3 thermal zones; from now on referred to as S5.
- A Sony Xperia Z5 based on a Snapdragon 810 SoC with Android 7.0 and 36 thermal zones, referred to as Z5.

If not otherwise specified, the two smartphones are placed in an air-conditioned server room with approximately 22°C and are connected to the power outlet. To generate our datasets, we use a measurement setup based on the ExOT (see Miedl et al. [29]). ExOT provides building blocks for measurement applications, as well as an experiment execution and analysis flow. To read the thermal zones, we employ a sink application which reads the corresponding sysfs files[5].

In addition, for the evaluation purposes, the sink application determines the current foreground application, i. e., the ground truth. Note that this just implemented for determining the ground truth and is not part of the attack scenario at all. The foreground application is determined by querying the usage stats or activity manager, depending on the Android build. As these methods require elevated privilege levels, in a real attack the sink application is not able to determine the foreground application.

[5] On the most Unix based systems the thermal zone nodes can be accessed via the `sysfs` where `$i` is the respective thermal zone number: `/sys/devices/virtual/thermal/thermal_zone$i/temp`

Table 1 Used applications and associated labels during the performance evaluation.

Label	Application Name	Description
VM	AnTuTu	Benchmark Suite
CL	Dropbox	Cloud storage application
DO	Document Viewer	Standard document viewer
GA	Angry Birds Rio	Game
SM	Facebook	Social Media client
IM	Wire	Instant messaging service
WB	Chrome Browser	Web browser
MA	Gmail	E-mail client
LO	Google Maps	Location services
VI	YouTube	Video streaming service
HO	Launcher/Home	Android system
-	Blank/Unknown	Unknown application

The sink application is implemented as an Android background service and is configured to conduct a measurement every 1 ms. However, due to the Android scheduling policy, the sampling period is longer and fluctuates. This results in an average sampling period of approximately 30 ms on S5 and 46 ms on Z5. To get equally-spaced samples for further data processing, we re-sample all thermal traces with a sampling period of 40 ms.

To minimise the data processing overhead, we embed our analysis into the existing ExOT framework to take advantage of the data processing stack. We randomly choose ambient thermal offset between −35°C and 35°C and allow multiple ambient temperature changes per batch. Table 1 outlines all possible foreground applications and the associated labels, if not defined otherwise. We selected those applications since they cover the most common use cases of a current smartphone.

7.1 Thermal parameters

For applying a thermal side channel attack according to Figure 2, we need to train the corresponding sequence model, see Figure 3. To be able to apply the data augmentation scheme as described in section 4, we need to characterise the device and determine the necessary thermal parameters, namely, the internal thermal, ambient heating and ambient cooling coefficients.

We also use the thermal coefficients to select the thermal zones which show high thermal dynamics, i. e., potentially carry useful information. Initial measurements on Z5 show that 24 of the 36 thermal zones provide usable thermal measurements. The other thermal zones either only provide 0°C outputs or are not affected by any application execution. Furthermore, on S5, we exclude the battery sensor as it does not show any application-dependent thermal variations. All parameters are outlined in Figure 9.

To determine the thermal coefficients β_z, we place the target devices in a testbed [38, Chapter 3], which allows us to control the temperature in a range of approximately 15°C to 50°C. For each ambient temperature, we conduct two measurements where we keep the device idle for 3 minutes and then execute a CPU benchmark[6] for either 70 or 130 seconds. The execution of the benchmark heats up the device, and we observe the cooling-off phase for 3.5 minutes, as illustrated in Figure 10.

[6] CPU Throttling Test (`https://play.google.com/store/apps/details?id=skynet.cputhrottlingtest`)

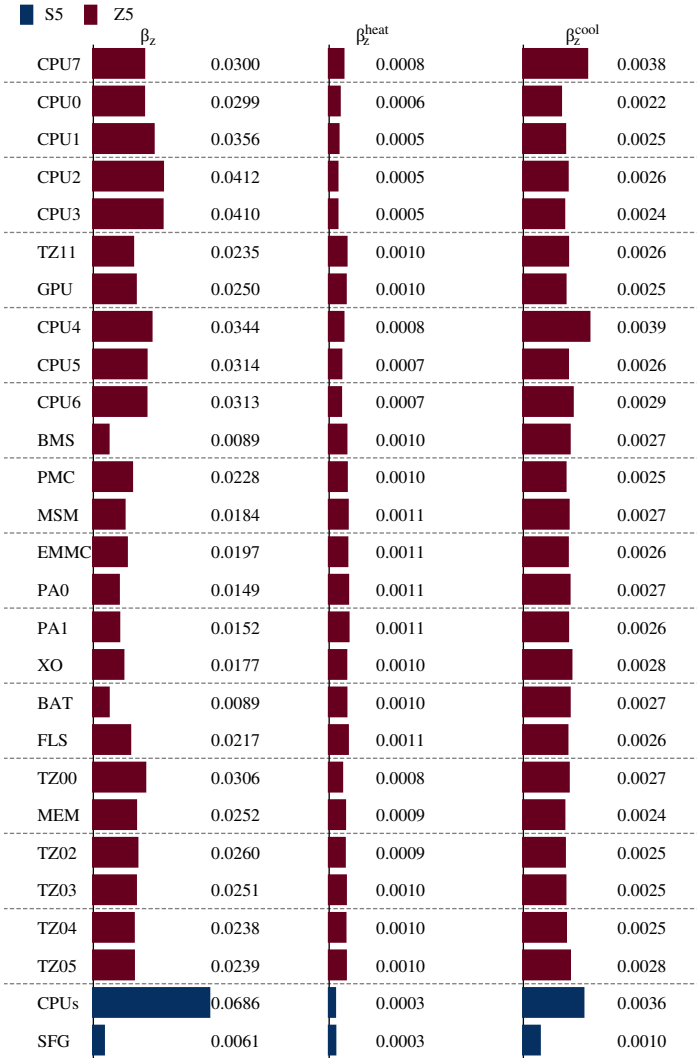


Figure 9 Thermal coefficients for internal (β_z), increasing (β_z^{heat}) and decreasing ambient temperatures.

To determine the ambient heating and cooling coefficients, we simply move the idle phones to different locations in- and outside of an office building. From the resulting thermal trace, we isolate the resulting thermal changes and determine β_z^{heat} and β_z^{cool}. The results in Figure 9 illustrate that the ambient thermal coefficients β_z^{heat} and β_z^{cool} are significantly lower than the respective β_z, on both platforms. However, due to the length of our temperature observations, the long-term influence of varying ambient temperatures cannot be neglected.

The thermal coefficients illustrated in Figure 9 show that the CPUs have the highest thermal dynamics on both smartphones S5 and Z5. Furthermore, while the internal thermal coefficients β_z for the cores are high compared to other thermal zones, the influence of increasing ambient temperatures is rather low, as indicated by the comparison of β_z^{heat} for different zones. Therefore, we will only use the eight CPU thermal zone readings on S5 and the single CPU cluster reading on Z5 as input for the application detection.

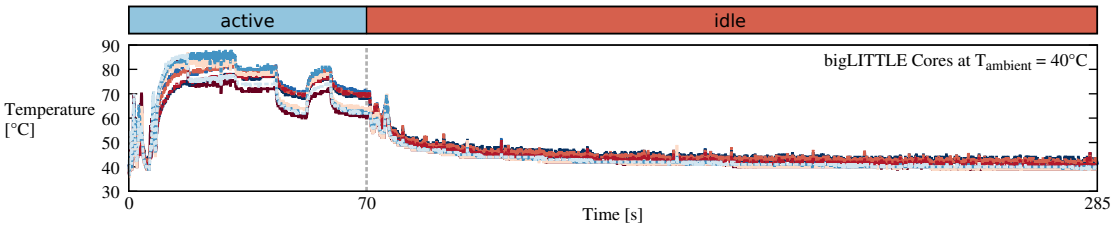


Figure 10 The device is heated up by an active application and we derive the thermal coefficients from the measurements taken when the device is idle and cools down.

7.2 Preparing thermal profiles for data augmentation

In order to generate the thermal profiles $T^a(t, z)$ required by the data augmentation scheme, see Figure 3, we need user inputs to interact with the selected applications. We use the RepetiTouch Pro application[7] to record user inputs. RepetiTouch Pro also allows us to replay recorded user inputs to generate multiple thermal profiles for one application usage. In our experimental setup, the user inputs are collected by a single user, who executes 49 different use cases. Each use case defines the usage of one application in a specific manner. By defining multiple use cases for each application, we ensure that the model learns the thermal profile of an application rather than one specific use case. In order to increase the pool of thermal profiles to feed to the data augmentation scheme, we record 10 traces per use case, i. e., 10 thermal profiles per use case.

The collected thermal profile traces can contain labels which we do not specify in Table 1. This is caused by dynamic application content, for example, advertisement pop-ups. As the execution of recorded user inputs with RepetiTouch is static, such dynamic application content causes deviations from the defined use case and results in unknown application labels. Depending on the evaluation scenario, we remove parts with unknown labels and use data augmentation to ensure there are no temperature discontinuities in the thermal profiles.

Moreover, we define that the data augmentation scheme uses thermal profile snippets with a length of at least 30s. Hence, our scheme is limited to a minimal use of a single application of 30 seconds. Shorter usages of applications are not considered at this stage.

8 Performance evaluation

In this section, we evaluate the performance of the thermal side channel attack based on the training and test datasets defined in subsection 7.2. We divide our performance evaluation into multiple scenarios. For each scenario, we generate 28 training and 7 test batches, with a batch length of 3 hours and 45 minutes (13500 s). The augmentation scheme uses 8 thermal profiles of each use case to generate the training dataset and the remaining 2 thermal profiles of each use case as a base for the test dataset.

No Augmentation Scenario. In the "No Aug" scenario, we train the network using the raw data we have collected without using the proposed augmentation technique. Thermal sequences that are shorter than the specified model input are zero-padded, and we use zero weights so that the training is not affected by the padding. This results in 325 training batches, one for each application use case. As a test, we generate 7 batches by simply concatenating recorded lab traces without augmenting the ambient temperature and the dynamic offset.

[7] `https://play.google.com/store/apps/details?id=com.cygery.repetitouch.pro`

Whitebox Scenario. The training dataset for the "Whitebox" scenario, generated using the data augmentation scheme, only contains known labels. This means we remove all portions with unknown labels from the datasets as described in subsection 7.2.

New Apps Scenario. In the "New Apps" scenario, we evaluate the performance of the model if a new application is added to the device after training using augmented data. In contrast to the "Whitebox" scenario, we keep all unknown labels in the training dataset. Furthermore, we record thermal profiles using the AndroBench[8] application and add it to the test data. Instead of introducing new labels for applications not defined in Table 1, the unknown ("-") label is used.

8.1 Expressiveness of the performance metrics

First, we intend to validate whether the chosen performance metrics are sufficiently expressive. To this end, we perform a manual evaluation by means of a visual inspection of a sample of the "New Apps" scenario, shown in Figure 11, and compare it to the performance metrics illustrated in Figure 12.

The per-time-step accuracy outlined in Figure 12 indicates that the models perform better with data from Z5 than S5, which is also supported by the trace illustrated in Figure 11. However, the relative Levenshtein distance is quite high for both platforms, as the experiments yield 0.83 for S5 and 0.70 for Z5. This indicates that the model misclassifies short thermal patterns, which are not compensated by the majority filtering. As the durations of the misclassified trace intervals are very short, they cause a higher relative Levenshtein distance while still yielding a high per-time-step accuracy. We assume that this behaviour is mainly caused by the fact that thermal patterns occurring during the execution of an application are very similar to the thermal pattern when starting other applications and, therefore, are misclassified. A possible solution to this issue could be to increase the long-term-memory of the model, to increase the amount of temporal context information that is taken into account by the model when placing the labels. The majority filter size could also be increased, but this would also increase the minimal detectable application execution length. This example shows that neither the per-time-step-accuracy nor the relative Levenshtein distance on its own are expressive regarding the performance of the model. Only when combined can these metrics be used to assess the performance of the models reliably.

The average temporal errors, shown in Figure 12 for the two platforms, S5 and Z5, are significantly lower than the length of the majority filtering window of 25 s. This indicates that the temporal error of the labelling models is mainly due to label misclassifications. This assumption is also supported by the labelling sequences illustrated in Figure 11.

8.2 Effectiveness of the data augmentation scheme

The experimental results depicted in Figure 12 show that training without augmentation ("No Aug") is not able to learn a proper sequence model, in contrast to the "Whitebox" and the "New Apps" scenario.

To provide a more detailed experimental analysis of the effectiveness of the data augmentation scheme, we run an experiment where we generate the training data with a reduced set of raw thermal profiles, i. e., we use 2 instead of 8 thermal profiles per use case as a basis for the training dataset. Figure 13 illustrates the results for the "New Apps" scenario with the normal and a reduced raw thermal profile set as used for generating the training dataset. For Z5, the performance for the reduced training data scenario decreases. For S5, however, training fails completely. We assume that these two effects are caused by the following factors:

[8] `https://play.google.com/store/apps/details?id=com.andromeda.androbench2`

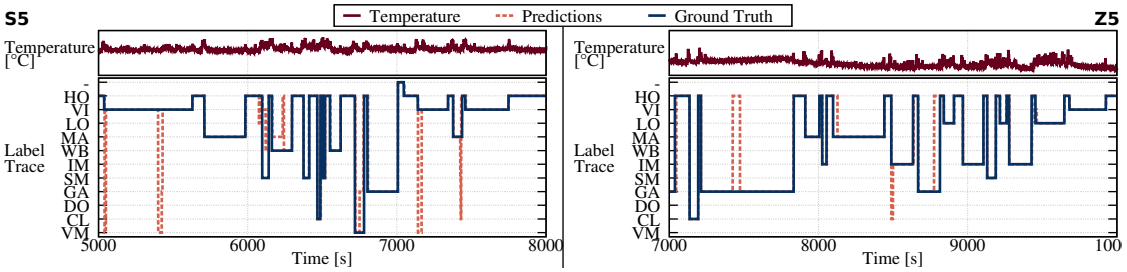


Figure 11 Traces indicate a higher amount of labelling errors on S5 than on Z5.

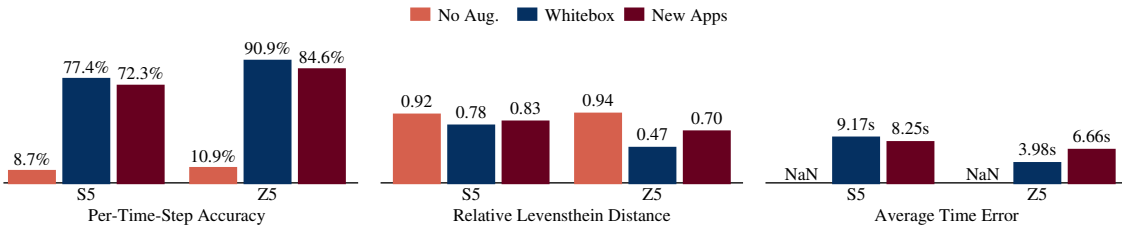


Figure 12 Training the model without augmented data is not possible. Models trained with augmented data perform well, but adding a new application cause performance degradation.

- The performance for Z5 degrades as the model does not generalise well. The augmentation scheme is not capable of generating a training data set with sufficient variance to allow the model to generalise well based on two very similar thermal profiles for each use case.
- For S5, training of the sequence model is not possible anymore, as the thermal profiles chosen for the training set are not representative. This might be caused by the fact that the measurement of the thermal profiles contains more measurement artefacts. Therefore, the smaller the set of thermal profiles, the more important their quality.

In sum, we can state that while the proposed augmentation scheme helps to generate datasets which allow for the training of the sequence model, there are some caveats that need to be considered. The quality of the training dataset highly depends on the quality of the thermal profiles. However, as there is the risk of measurement artefacts, data cleansing is necessary, as the quality of the training and test data highly depends on the quality of the initial raw data. Unfortunately, data cleansing is a process which can hardly be automated and, to some extent, will always have to be done manually.

8.3 Performance on different devices

The results illustrated in Figure 12 show that in general models trained for Z5 outperform the models for S5. While we mentioned in the previous subsection that the quality of the thermal profiles collected on S5 seem to be less representative, we also assume that the lower performance on S5 is caused by the lack of thermal information that we can collect. On Z5, we can use 8 sensor reading for 8 cores, while on S5 we can only get 1 reading for all 8 cores. Therefore, the amount of thermal information that we can extract is lower on S5, which lowers the performance of the sequence model.

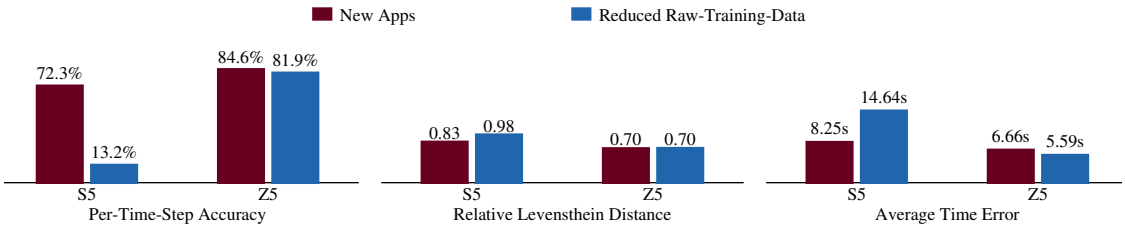


Figure 13 Reducing the base dataset of thermal profiles for the data augmentation causes a slight performance drop for Z5, but has a substantial impact on the performance for data from S5. This is caused by the dependency of our data augmentation scheme on the amount and quality of the thermal profiles available for data generation.

8.4 Influence of new applications

To assess the influence of a new application in the test dataset, we compare the performance metrics for the "Whitebox" and the new app scenario illustrated in Figure 12. The metrics suggest that the model performance suffers when the test data contains applications labelled as unknown, which are not present in the training dataset. A detailed analysis of the labelling traces shows that the accuracy degradation of the per-time-step accuracy from the "Whitebox" to "New Apps" scenario is approximately the amount of unknown label in the test dataset. Therefore, we conclude that our models are not capable to properly label new applications with the unknown ("-") label.

The models react as expected and simply label the new application with the application label, which has the most similar thermal profile. This issue can be addressed by either implementing unsupervised online learning to update the model constantly or offline re-training of the model. However, these extensions are left for future work.

8.5 Single application detection

For the final laboratory test, we evaluate the performance of the model when it only has to detect whether a specific application is running or not. Therefore, in the training set, we only have two labels

(i) the targeted application, or
(ii) unknown.

Figure 14 illustrates the performance metrics for the different applications, i. e., each case corresponds to a completely new learning scenario where one of the applications is known and all other applications are labelled as unknown ("-").

Except for Chrome ("WB"), Google Maps ("MA") and YouTube ("VI"), on both platforms single applications are detected with an accuracy above 80%, sometimes even exceeding 90%. However, we also have to consider the relative Levenshtein distance, which is above 0.5 for many test-cases and also high average timing errors. We assume that the lower performance for some applications is caused by more ambiguous thermal profiles. For example, for YouTube ("VI"), the thermal profile is very different depending on whether a video is played or the application is only opened and the search function is used, without video playback. Therefore, we conclude that it might also be useful to introduce multiple labels per application to differentiate, for example, different operations like menu and video replay.

Training a model for single application detection is harder than for the multilabel case. This is caused by the fact that the real data and the training data might be very different. While we configured our augmentation scheme to generate training data sets to contain many occurrences of the target application, in real life, a target application might not be used for a long time. However,

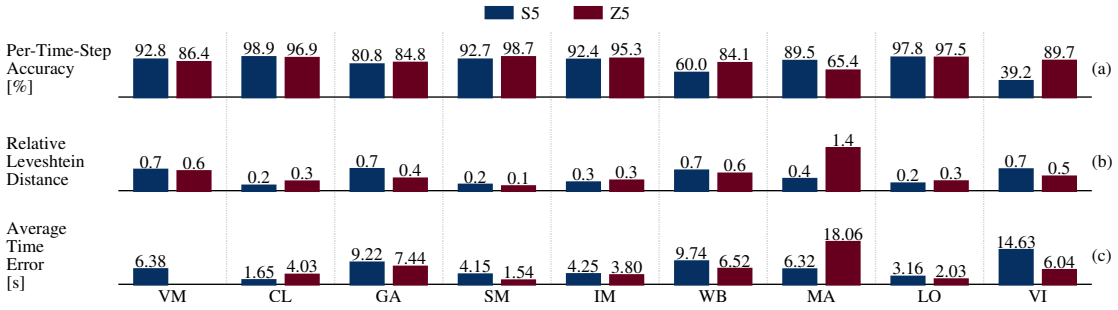


Figure 14 Performance metrics for the detection of a single application. While the per-time-step accuracy is very high for most applications, the relative Levenshtein distance varies between 0.1 to 0.6. This indicates that the applications with a high relative Levenshtein distance have a less unique thermal profile. Furthermore, the time errors indicate that the labels are placed with an almost perfect temporal accuracy, as they are considerably smaller than the majority voting filter window of 25 s.

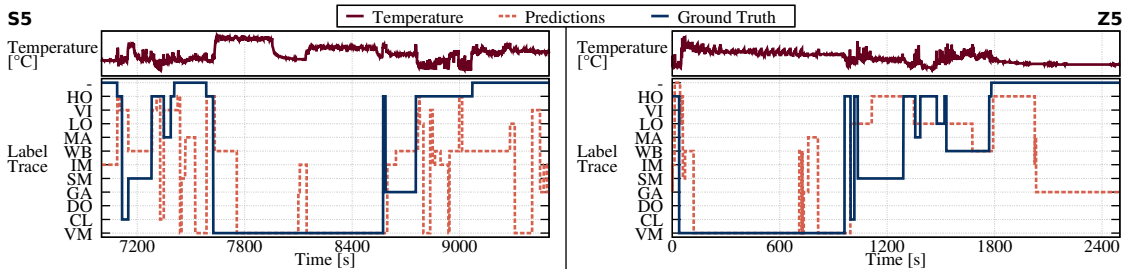


Figure 15 The real-world traces for both smartphones, S5 and Z5. The short temporal snippets validate the performance metrics and illustrate that the models are not able to correctly label the thermal trace.

if the target application does not appear often enough in the training data, the model will not be able to learn the thermal profile of the application. On the other extreme, if the target application thermal profile is inserted too often and with an unintended periodicity, the model might learn the periodicity in which the target application appears, rather than the thermal profile. Therefore, a model needs to be trained carefully to learn the target application thermal profile without expecting this thermal profile to occur regularly.

If trained properly and when considering the observations from subsection 8.4, a model for single application detection might be more robust in a real deployment. Let us consider that the multidimensional feature space used by the RNN to classify the thermal traces is a high-dimensional Euclidean space. In such a case, the similarity of two thermal profiles can be illustrated by the distance between the two corresponding points in the feature space. The space mapped to the target application label will be small compared to the space which maps to all the other unknown applications. Therefore, if a new application thermal profile is presented to the model, its representation will most likely be mapped to the unknown label. Hence, the model performance will not degrade, i. e., the model is robust against new applications.

8.6 Real-world applicability

In addition to the lab generated data, we also collect a real-world trace: the same user who recorded the inputs for the laboratory setup carries each smartphone for a day. The user freely runs the applications available on the smartphones to imitate normal behaviour. As no data is recorded when the smartphone is locked, this results in trace lengths of

(i) 3h (10800 s) for S5, and
(ii) 4.5h (16200 s) for Z5.

The models trained with the "New Apps" scenario training set achieved a per-time-step accuracy of less than 20%. This means that the models are not able to detect any applications in the real-world thermal trace correctly, as outlined in the example illustrated in Figure 15. Also, models for the binary use cases were not capable of detecting the respective target application in the real-world trace.

The real world data shows more short term variability than the laboratory data. Causes for this are the variable temperature influences from the environment and the position of the phone, e.g. how the phone is held and operated. Furthermore, the laboratory traces were collected with a minimum amount of applications running in the background, while during the real world data more applications might be active and disturb the thermal profile of a foreground application. Therefore, the significance and shape of the thermal features that the model uses change, which makes it harder for the model to correctly infer the right application based on thermal data. In addition, the user interaction pattern in the real-world experiment could have deviated significantly from the typical usage patterns recorded for training. This would have resulted in significantly different thermal profiles and consequently, higher error.

8.7 Future directions

While the results based on laboratory data are promising and clearly show the threat potential of the thermal side channel, the tests using data collected outside of the laboratory illustrate that there are still challenges to overcome to implement the thermal side channel attack in a real environment. As the sequence model requires a large amount of data for training, we do not consider manual data collection outside of a laboratory setup a viable alternative to a data augmentation scheme. However, the data augmentation scheme needs to be refined to resemble real-world data more closely. This includes, for example, influences on the temperature of the smartphone when it is held in the hand compared to sitting on a table. In addition, the current scheme does not take into account that the set of applications running in the background changes under normal use. This influences the scheduling and core pinning of the foreground application and adds noise to the profile, whereas in our evaluation all thermal profiles for training are collected from smartphones running the same set of background services.

Furthermore, a study evaluating the relation between severity of the side channel and the amount of available thermal information would allow more insights in possible mitigation strategies. Sparse spatial information, less available sensors, or a low temporal resolution of the measurements might drastically reduce the possibility to mount the thermal side channel attack, while still providing enough thermal information to the power management system.

Another possible direction for exploration are server racks or laptops. Due to the lower mobility we expect the environmental influences to be lower. However, the cooling systems and the high utilisation of those systems might make a thermal side channel attack, as presented in this work, very challenging.

In future work, the online scenario also needs to be evaluated. This requires that the sequence model is optimised for the attacked platforms, to minimise the computation and memory footprint. Otherwise, the attack is easy to detect or it might not even be possible to deploy it, due to the limited hardware capabilities of the smartphones. Moreover, evaluating online learning methods that would allow the model to predict applications it has not seen during the initial training would further increase the capabilities of the thermal side channel attack.

9 Concluding remarks

In this work, we presented a data leak based on the measurement of the temperature of a mobile device using its internal sensors. We showed that it is possible to determine which applications were executed at what time. This kind of information can be acquired without the knowledge of the user and may pose a serious security and privacy violation.

We showed how to generate a fitting dataset for training a model based on real-world measurements but without the need for an extensive measurement campaign. Furthermore, we explained in detail how to build and train this time-sequence model using Convolutional-Neural-Network (CNN), Long Short-Term Memory (LSTM) and label trace filtering. In addition, we outlined an extensive laboratory study based on data from two smartphones, a Samsung Galaxy S5 and a Sony Xperia Z5. The results of the laboratory results are promising, with per-time-accuracy of up to 90% for a scenario with 11 different application labels. However, tests using data recorded outside of the laboratory setup revealed that the data augmentation scheme is not sophisticated enough to use laboratory data to generate training datasets that resemble outside use.

Lastly, we showed that it is possible to misuse this thermal information to mount a thermal side channel attack. Because a thermal side channel attack violates security and privacy constraints, action needs to be taken to mitigate this data leak before it can become a real threat.

References

1 Davide B. Bartolini, Philipp Miedl, and Lothar Thiele. On the Capacity of Thermal Covert Channels in Multicores. In *Proceedings of the Eleventh European Conference on Computer Systems*, EuroSys '16, pages 24:1–24:16. ACM, 2016. doi:10.1145/2901318.2901322.

2 J. Brouchier, T. Kean, C. Marsh, and D. Naccache. Temperature Attacks. *IEEE Security and Privacy*, 7(2):79–82, March 2009. doi:10.1109/MSP.2009.54.

3 Julien Brouchier, Nora Dabbous, Tom Kean, Carol Marsh, and David Naccache. Thermocommunication. Cryptology ePrint Archive, Report 2009/002, 2009. URL: https://eprint.iacr.org/2009/002.

4 Anna L Buczak and Erhan Guven. A survey of data mining and machine learning methods for cyber security intrusion detection. *IEEE Communications Surveys & Tutorials*, 18(2):1153–1176, 2016. doi:10.1109/COMST.2015.2494502.

5 Hong Cao and Miao Lin. Mining smartphone data for app usage prediction and recommendations: A survey. *Pervasive and Mobile Computing*, 37:1–22, 2017. doi:10.1016/j.pmcj.2017.01.007.

6 P Dadvar and K Skadron. Potential thermal security risks. In *Semiconductor Thermal Measurement and Management Symposium, 2005 IEEE Twenty First Annual IEEE*, pages 229–234, 2005. doi:10.1109/STHERM.2005.1412184.

7 Dmitry Evtyushkin and Dmitry Ponomarev. Covert channels through random number generator: Mechanisms, capacity estimation and mitigations. In *Proceedings of the 2016 ACM SIGSAC Conference on Computer and Communications Security*, CCS '16, pages 843–857. Association for Computing Machinery, 2016. doi:10.1145/2976749.2978374.

8 Dmitry Evtyushkin, Dmitry Ponomarev, and Nael Abu-Ghazaleh. Understanding and Mitigating Covert Channels Through Branch Predictors. *ACM Transactions on Architecture and Code Optimization (TACO)*, 13(1), March 2016. doi:10.1145/2870636.

9 Ian Goodfellow, Yoshua Bengio, and Aaron Courville. *Deep Learning*. MIT Press, 2016. URL: http://www.deeplearningbook.org.

10 Johannes Götzfried, Moritz Eckert, Sebastian Schinzel, and Tilo Müller. Cache Attacks on Intel SGX. In *Proceedings of the 10th European Workshop on Systems Security*, EuroSec'17. Association for Computing Machinery, 2017. doi:10.1145/3065913.3065915.

11 Michael C Grace, Yajin Zhou, Zhi Wang, and Xuxian Jiang. Systematic detection of capability leaks in stock android smartphones. In *NDSS*, volume 14, page 19, 2012.

12 Alex Graves. *Supervised sequence labelling*, pages 5–13. Springer Berlin Heidelberg, Berlin, Heidelberg, 2012. doi:10.1007/978-3-642-24797-2_2.

13 Mordechai Guri, Matan Monitz, Yisroel Mirski, and Yuval Elovici. BitWhisper: Covert Signaling Channel between Air-Gapped Computers Using Thermal Manipulations. In *Proceedings of the 2015 IEEE 28th Computer Security Foundations Symposium*, CSF '15, pages 276–289, USA, 2015. doi:10.1109/CSF.2015.26.

14 Sepp Hochreiter and Jürgen Schmidhuber. Long Short-Term Memory. *Neural Comput.*, 9(8):1735–1780, November 1997. doi:10.1162/neco.1997.9.8.1735.

15 Michael Hutter and Jörn-Marc Schmidt. *The Temperature Side Channel and Heating Fault Attacks*, pages 219–235. Springer International Publishing, Cham, 2014. doi:10.1007/978-3-319-08302-5_15.

16 T. Iakymchuk, M. Nikodem, and K. Kepa. Temperature-based covert channel in FPGA systems. In *Reconfigurable Communication-centric*

Systems-on-Chip (ReCoSoC), 2011 6th International Workshop on, pages 1–7, June 2011. doi:10.1109/ReCoSoC.2011.5981510.

17 Mohammad A. Islam, Shaolei Ren, and Adam Wierman. Exploiting a Thermal Side Channel for Power Attacks in Multi-Tenant Data Centers. In *Proceedings of the 2017 ACM SIGSAC Conference on Computer and Communications Security*, CCS '17, pages 1079–1094. Association for Computing Machinery, 2017. doi:10.1145/3133956.3133994.

18 Paul Kocher, Daniel Genkin, Daniel Gruss, Werner Haas, Mike Hamburg, Moritz Lipp, Stefan Mangard, Thomas Prescher, Michael Schwarz, and Yuval Yarom. Spectre attacks: Exploiting speculative execution. *arXiv preprint arXiv:1801.01203*, 2018. URL: https://spectreattack.com/.

19 Butler W. Lampson. A Note on the Confinement Problem. *Commun. ACM*, 16(10):613–615, October 1973. doi:10.1145/362375.362389.

20 Moritz Lipp, Daniel Gruss, Raphael Spreitzer, Clémentine Maurice, and Stefan Mangard. ARMageddon: Cache Attacks on Mobile Devices. In *Proceedings of the 25th USENIX Conference on Security Symposium*, SEC'16, pages 549–564. USENIX Association, 2016. doi:10.5555/3241094.3241138.

21 Moritz Lipp, Michael Schwarz, Daniel Gruss, Thomas Prescher, Werner Haas, Stefan Mangard, Paul Kocher, Daniel Genkin, Yuval Yarom, and Mike Hamburg. Meltdown. *arXiv preprint arXiv:1801.01207*, 2018. URL: https://spectreattack.com/.

22 Carol Marsh and David McLaren. Poster: Temperature Side Channels. In *In the Proceedings of the 9th International Workshop on Cryptographic Hardware and Embedded Systems (CHES), 2007*, 2007.

23 Ramya Jayaram Masti, Devendra Rai, Aanjhan Ranganathan, Christian Müller, Lothar Thiele, and Srdjan Capkun. Thermal Covert Channels on Multi-core Platforms. In *24th USENIX Security Symposium (USENIX Security 15)*, pages 865–880, Washington, D.C., August 2015. USENIX Association. doi:10.5555/2831143.2831198.

24 Clémentine Maurice, Manuel Weber, Michael Schwarz, Lukas Giner, Daniel Gruss, Carlo Alberto Boano, Stefan Mangard, and Kay Römer. Hello from the other side: SSH over robust cache covert channels in the cloud. *NDSS, San Diego, CA, US*, 2017. URL: https://cmaurice.fr/pdf/ndss17_maurice.pdf.

25 Matthias Meyer, Samuel Weber, Jan Beutel, and Lothar Thiele. Systematic identification of external influences in multi-year microseismic recordings using convolutional neural networks. *Earth Surface Dynamics*, 7(1):171–190, 2019. doi:10.5194/esurf-7-171-2019.

26 Yan Michalevsky, Gabi Nakibly, Gunaa Arumugam Veerapandian, Dan Boneh, and Gabi Nakibly. PowerSpy: Location Tracking Using Mobile Device Power Analysis. In *24th USENIX Security Symposium (USENIX Security 15)*, pages 785–800, Washington, D.C., August 2015. USENIX Association. URL: https://www.usenix.org/conference/usenixsecurity15/technical-sessions/presentation/michalevsky.

27 Philipp Miedl, Xiaoxi He, Matthias Meyer, Davide Basilio Bartolini, and Lothar Thiele. Frequency Scaling as a Security Threat on Multicore Systems. *IEEE Transactions on Computer-Aided Design of Integrated Circuits and Systems*, 37(11):2497–2508, November 2018. doi:10.1109/TCAD.2018.2857038.

28 Philipp Miedl, Bruno Klopott, and Lothar Thiele. ExOT Website, March 2020. URL: https://www.exot.ethz.ch/.

29 Philipp Miedl, Bruno Klopott, and Lothar Thiele. Increased reproducibility and comparability of data leak evaluations using ExOT. In *2020 Design, Automation & Test in Europe Conference & Exhibition (DATE)*. IEEE, 2020. doi:10.3929/ethz-b-000377986.

30 Philipp Miedl and Lothar Thiele. The Security Risks of Power Measurements in Multicores. In *Proceedings of the 33rd Annual ACM Symposium on Applied Computing*, SAC '18, pages 1585–1592. Association for Computing Machinery, 2018. doi:10.1145/3167132.3167301.

31 Steven J. Murdoch. Hot or Not: Revealing Hidden Services by Their Clock Skew. In *Proceedings of the 13th ACM Conference on Computer and Communications Security*, CCS '06, pages 27–36. Association for Computing Machinery, 2006. doi:10.1145/1180405.1180410.

32 Naser Peiravian and Xingquan Zhu. Machine learning for android malware detection using permission and api calls. In *Proceedings of the 2013 IEEE 25th International Conference on Tools with Artificial Intelligence*, ICTAI '13, pages 300–305, USA, 2013. IEEE Computer Society. doi:10.1109/ICTAI.2013.53.

33 Danny Philippe-Jankovic and Tanveer A Zia. Breaking VM Isolation-An In-Depth Look into the Cross VM Flush Reload Cache Timing Attack. *International Journal of Computer Science and Network Security (IJCSNS)*, 17(2):181, 2017. URL: https://researchoutput.csu.edu.au/en/publications/breaking-vm-isolation-an-in-depth-look-into-the-cross-flush-reloa-2.

34 Sashank J. Reddi, Satyen Kale, and Sanjiv Kumar. On the Convergence of Adam and Beyond, 2019. arXiv:1904.09237.

35 Thomas Ristenpart, Eran Tromer, Hovav Shacham, and Stefan Savage. Hey, you, get off of my cloud: exploring information leakage in third-party compute clouds. In *Proceedings of the 16th ACM conference on Computer and communications security*, CCS '09, pages 199–212. Association for Computing Machinery, 2009. doi:10.1145/1653662.1653687.

36 Hong Rong, Huimei Wang, Jian Liu, Xiaochen Zhang, and Ming Xian. WindTalker: An Efficient and Robust Protocol of Cloud Covert Channel Based on Memory Deduplication. In *Proceedings of the 2015 IEEE Fifth International Conference on Big Data and Cloud Computing*, BDCLOUD '15, pages 68–75, USA, 2015. IEEE Computer Society. doi:10.1109/BDCloud.2015.12.

37 Stan Salvador and Philip Chan. Toward Accurate Dynamic Time Warping in Linear Time and Space. *Intell. Data Anal.*, 11(5):561–580, October 2007. doi:10.5555/1367985.1367993.

38 Lukas Sigrist. *Design and Instrumentation of Environment-Powered Systems*. PhD thesis, ETH Zurich, 2020.

39 Raphael Spreitzer, Simone Griesmayr, Thomas Korak, and Stefan Mangard. Exploiting data-usage statistics for website fingerprinting attacks on android. In *Proceedings of the 9th ACM Conference on Security & Privacy in Wireless and Mobile Networks*, WiSec '16, pages 49–60. Association for Computing Machinery, 2016. doi:10.1145/2939918.2939922.

40 Shanquan Tian and Jakub Szefer. Temporal Thermal Covert Channels in Cloud FPGAs. In *Proceedings of the 2019 ACM/SIGDA International Symposium on Field-Programmable Gate Arrays*, FPGA '19, pages 298–303. Association for Computing Machinery, 2019. doi:10.1145/3289602.3293920.

41 Tijmen Tieleman and Geoffrey Hinton. Lecture 6.5-rmsprop: Divide the gradient by a running average of its recent magnitude. *COURSERA: Neural networks for machine learning*, 4(2):26–31, 2012.

42 Xu, Yunjing and Bailey, Michael and Jahanian, Farnam and Joshi, Kaustubh and Hiltunen, Matti and Schlichting, Richard. An Exploration of L2 Cache Covert Channels in Virtualized Environments. In *Proceedings of the 3rd ACM Workshop on Cloud Computing Security Workshop*, CCSW '11, pages 29–40. Association for Computing Machinery, 2011. doi:10.1145/2046660.2046670.

43 S. Zander, P. Branch, and G. Armitage. Capacity of Temperature-Based Covert Channels. *Communications Letters, IEEE*, 15(1):82–84, 2011. doi:10.1109/LCOMM.2010.110310.101334.

44 Sebastian Zander and Steven J. Murdoch. An Improved Clock-skew Measurement Technique for Revealing Hidden Services. In *Proceedings of the 17th USENIX Security Symposium*, SS'08, pages 211–226. USENIX Association, 2008. doi:10.5555/1496711.1496726,.