






## Article

# Chidroid: A Mobile Android Application for Log Collection and Security Analysis in Healthcare and IoMT

Stylianos Karagiannis <sup>1,2,\*</sup>, Luís Landeiro Ribeiro <sup>1,†</sup>, Christoforos Ntantogian <sup>2</sup>, Emmanouil Magkos <sup>2</sup>  
and Luís Miguel Campos <sup>1</sup>

<sup>1</sup> PDM&FC, R. Fradesso da Silveira, 4-1B, 1300-609 Lisboa, Portugal

<sup>2</sup> Department of Informatics, Ionian University, Plateia Tsirigoti 7, 49100 Corfu, Greece

\* Correspondence: stylianos.karagiannis@pdmfc.com or skaragiannis@ionio.gr

† These authors contributed equally to this work.

**Abstract:** The Internet of Medical Things (IoMT) is a growing trend that has led to the use of connected devices, known as the Internet of Health. The healthcare domain has been a target of cyberattacks, especially with a large number of IoMT devices connected to hospital networks. This factor could allow attackers to access patients' personal health information (PHI). This research paper proposes Chidroid, an innovative mobile Android application that can retrieve, collect, and distribute logs from smart healthcare devices. The proposed approach enables the creation of datasets, allowing non-structured data to be parsed into semi-structured or structured data that can be used for machine learning and deep learning, and the proposed approach can serve as a universal policy-based tool to examine and analyse security issues in most recent Android versions by distributing logs for analysis. The validation tests demonstrated that the application could retrieve logs and system metrics from various assets and devices in an efficient manner. The collected logs can provide visibility into the device's activities and help to detect and mitigate potential security risks. This research introduces a way to perform a security analysis on Android devices that uses minimal system resources and reduces battery consumption by pushing the analysis stage to the edge.

**Keywords:** android security; log analysis; IoMT security; IoT security; artificial intelligence; IoMT



**Citation:** Karagiannis, S.; Ribeiro, L.L.; Ntantogian, C.; Magkos, E.; Campos, L.M. Chidroid: A Mobile Android Application for Log Collection and Security Analysis in Healthcare and IoMT. *Appl. Sci.* **2023**, *13*, 3061. <https://doi.org/10.3390/app13053061>

Academic Editors: Stefano Silvestri and Francesco Gargiulo

Received: 18 January 2023

Revised: 16 February 2023

Accepted: 21 February 2023

Published: 27 February 2023



**Copyright:** © 2023 by the authors. Licensee MDPI, Basel, Switzerland. This article is an open access article distributed under the terms and conditions of the Creative Commons Attribution (CC BY) license (<https://creativecommons.org/licenses/by/4.0/>).

## 1. Introduction

The world currently is heavily dependent on the Internet of Things (IoT) for our daily lives [1]. The IoT is a rapidly growing field and is expected to expand in the coming years. This growth is likely driven by various factors, including the increasing affordability and availability of smart devices and the growing demand for connected products and services.

Android is an operating system often used on the IoT, based on the Linux kernel, acquired initially and currently developed by Google, and was first released in 2008. The Android operating system supports the IoT, which means that it can be used to connect and control a wide range of smart devices. For example, an Android device can monitor and maintain smart home appliances, such as thermostats and lighting systems. In addition, many Android devices are equipped with sensors and other hardware to collect data from the environment and provide input to IoT applications. Overall, the combination of Android and the IoT offers many possibilities to create useful and innovative connected products and services [2–4]. Smartphones and other mobile devices are used for mobile health applications (mHealth applications) and recently have played an increasingly important role in the healthcare industry [5]. Android and mHealth applications are used for various purposes, including patient care, clinical decision support, exchange of health information, and remote patient monitoring [6]. In addition, Android devices are increasingly popular in the healthcare domain, but as they become more commonplace, so do their security and privacy risks.

According to ENISA and the report published in July 2022 (<https://www.enisa.europa.eu/news/ransomware-publicly-reported-incidents-are-only-the-tip-of-the-iceberg>), “Between May 2021 and June 2022 about ten terabytes of data were stolen each month by ransomware threat actors. 58.2% of the stolen data included employees’ personal data”. The study reports that the most-affected sectors include heavy industry, information services, government, and healthcare [7]. A recent relevant example of cyberattacks in the healthcare domain was the case of the Ryuk ransomware [8], as mentioned by ENISA’s Threat Landscape Report 2021 (ETL 2021) (<https://www.enisa.europa.eu/publications/enisa-threat-landscape-2021>, accessed 20 February 2023).

The healthcare domain has been a target of cyberattacks, especially with the growing use of connected devices known as the Internet of Medical Things (IoMT). Similarly, the IoT extends the threat landscape even further, with the most-common cyberattacks being denial-of-service (DoS) attacks or unauthorised access to sensitive health information [9]. In some cases, cyberattacks lead to the exposure of patient personal health information (PHI) [10,11]. To address these concerns, healthcare organisations should ensure that their IoT devices, infrastructure, and information systems are adequately secured. Furthermore, many IoMT devices are connected to hospital networks [12], a factor that could allow attackers to access patient health information or even disrupt medical care. There are several ways to perform security and privacy analysis in the IoMT. A common approach for conducting security assessments on Android is static analysis, which examines the source code of applications and operating systems for potential vulnerabilities. Another method is to use dynamic analysis to monitor the behaviour of these devices at runtime. Taking this into account, it is essential to devise mechanisms that retrieve logs and operating metrics to discover if an IoMT device is acting abnormally or has even been compromised [13–15].

The IoMT is characterised by the high interconnectivity of physical objects and devices embedded in sensors, software, and other technologies that enable them to collect and exchange data [16]. To protect patients’ security and privacy and safeguard their medical data, healthcare organisations must take measures to secure all mobile devices that have access to PHI [17]. This includes both employee-owned and corporate-owned devices. IoMT devices often collect sensitive health data, making them attractive targets for attackers [18,19]. Patients’ homes are also at risk if they have connected medical devices. According to research (ESET Threat Report T1 2022), the volume of IoT malware in the first six months of 2022 was higher than that recorded in the previous four years. This indicates a significant increase in the number of IoT devices targeted by attackers.

Considering the above, the security of the IoT and Android is essential, and research has been performed to enhance network authentication and mitigate attacks, contributing to the security of mobile health systems on 5G networks [20]. Therefore, it is critical to protect healthcare and IoMT devices from potential security threats, as these devices are used to handle sensitive medical and healthcare data. This research paper proposes Chidroid, an Android application that can retrieve, collect, and distribute logs for analysis on the cloud or on-premises servers.

Chidroid enables log collection and distribution to the cloud or other endpoints (servers, computers, Android devices) for analysis enabling edge computing. It can run on multiple edge devices, enabling edge computing and distributing the data to multiple endpoints for simultaneous management, allowing remote management of edge devices. Edge computing is a computational paradigm that brings computation and data storage closer to the network, where data are generated and used. Edge computing offers several advantages in the Internet of Medical Things (IoMT). Firstly, it enables the processing of large amounts of health-related data generated by various medical devices in real-time without overwhelming the back-end infrastructure. Secondly, edge computing can provide improved privacy and security as sensitive health information can be processed locally, reducing the risk of data breaches. Finally, edge computing can improve the overall system efficiency by reducing the amount of data transmitted to the cloud, thus reducing latency, bandwidth usage, and energy consumption.

### 1.1. Related Work

While there is a significant amount of research on the IoT and data and artificial intelligence (AI) in healthcare, it should be noted that the topics covered by these studies can vary greatly and do not directly relate to the context of this work. Some researchers study specific security or privacy issues, while others focus on reducing the cost of IoT sensors. To this end, we included the most-relevant research studies on Android security, the IoT, and edge computing with significant impacts on healthcare and eHealth. The selected related work is presented below and organised and summarised in a concise manner (Table 1).

**Table 1.** Table summarising the related work.

Topics	Paper Ref.	Description
Log Collection	[21]	Android security, log retrieval and analysis. Collection of system-generated logs at runtime and filtering for analysis.
	[22]	Log management tool to collect and display installation log files and track changes during the installation process.
Malware Analysis	[23]	Malware analysis and a framework combining machine learning and static analysis.
	[24]	Hybrid Android malware detection combining dynamic and static analysis.
	[25]	Framework for Android system call processing to detect malware early.
	[26]	BMD: bi-level malware detection using application programming interface (API) call sequences and co-evolutionary algorithm.
Permissions	[27]	Permission monitoring in Android security to categorise applications and use static analysis to detect malware.
	[28]	Permission classification as an information gain metric for improved detection accuracy.
Package Analysis	[29]	Review of the techniques combining static and dynamic analysis with studying Android applications.
	[30]	Static data flow analysis tool and library developed for the purpose of computing data flows in Android applications and Java programs.
	[31]	Python-based tool designed for reverse-engineering of Android applications by breaking down raw Android Package Kit (.apk) files.
	[32]	A static analysis framework for the security evaluation of Android apps.
HIDS	[33]	An HIDS system that runs on mobile devices and detects known and zero-day attacks using machine learning and statistical algorithms.
	[34]	A Review of system-call-based HIDS, including feature extraction, data mining algorithms, HIDS datasets, and applicability to embedded systems.

There are many research papers and studies on Android security, log retrieval, and analysis (Table 1). Some of these articles focused on specific aspects of Android security, such as the use of Logcat (<https://developer.android.com/studio/command-line/logcat>, accessed 20 February 2023) and Package Manager (<https://developer.android.com/reference/android/content/pm/PackageManager>, accessed 20 February 2023). In [21], the authors proposed an approach that involves collecting system-generated logs at runtime and filtering them for the application under analysis. These logs are then matched against generated signatures that account for the application's behaviour, such as information leakage, jailbreak attempts, privilege escalation attempts, and access to critical permissions. Another example is [22], where the authors proposed a log management tool to collect and display installation log files. These log files contain information about the installation process, such as the time and date of the installation, the user who installed the application, and the location of the installation files. Additionally, the tool can compare the contents of the log files before and after an application is installed, allowing users to see any changes that may have occurred during the installation process.

The field of malware analysis has seen the development of frameworks combining machine learning and static analysis ([23]) and hybrid Android malware detection approaches combining dynamic and static analysis [24]. One important aspect of malware analysis is the early detection of malicious activities to prevent harm to the system and its users.

A framework for system call processing has been proposed to address this issue, focusing on analysing system calls made by Android applications [25]. The bi-level malware detection method (BMD) was also introduced to improve detection accuracy by utilising API call sequences and a co-evolutionary algorithm [26]. The experiments performed on Android datasets showed that BMD outperforms existing state-of-the-art methods for malware detection.

Other articles focus on the role of permissions in Android security and permission monitoring to investigate access to the sensitive data and resources of the device [27]. In terms of permission monitoring, the researchers presented a method for classifying Android applications based on their access to private information and the abuse of this information. The method categorises applications into three types based on their access to private information. Then, a static analysis is used to detect Android malware utilising the information and permissions of the applications. In another study [28], the authors used the classification of permissions as an information gain metric to provide better detection accuracy.

In [29], the researchers proposed a technique that combines static and dynamic analysis to examine Android applications. This approach makes it possible to expand an application's method call graph (MCG) by capturing additional modules loaded at run-time and additional paths of execution concealed by reflection calls. The researchers also reported the analysis of recent state-of-the-art tools that should be taken into account, namely Flowdroid [30], AndroGuard [31], and Amandroid [32], among others. Other researchers [33] proposed using host-based intrusion detection systems (HIDSs) to run entirely on a mobile device without relying on a remote server. The application monitors the device by regularly sampling features that represent the overall use of the device's resources, such as the central processing unit (CPU), random access memory (RAM), battery, and others. The detection engine adopts machine learning and statistical algorithms to detect known and unprecedented (zero-day) attacks. Furthermore, in a review [34], various approaches were presented, including system-call-based host-based intrusion detection systems (HIDSs), examining various components. The research presented the extraction of features, the use of data-mining algorithms, the availability and utilisation of HIDS datasets, the implementation in current embedded systems, and future research directions.

Research on edge computing has also been considered and is not directly related to the current research. The literature includes studies on AI-based mechanisms for enhancing healthcare data security in the IoT-cloud and reducing the cost of IoT sensors [35]. A survey article [36] covered the edge architecture's security and privacy issues caused by heterogeneous device networking. Reference [37] provided a comprehensive analysis of data security and privacy in edge computing, including an overview of edge computing, security, and privacy requirements. Reference [38] explored the use of smart medical sensors and the IoT in healthcare and the limitations of cloud-based systems, highlighting the trend towards edge and fog computing as solutions. Reference [38] also provided an overview of the IoT in healthcare, covering early wearable-sensor-based health monitoring to recent advancements in fog/edge computing for smart health. Finally, Reference [38] comprehensively analysed data security and privacy in edge computing, including an overview of the fundamentals, challenges, mechanisms, state-of-the-art solutions, and proposed research directions.

In summary, prior research in the field of Android security has encompassed a range of approaches, including malware detection and security analysis techniques. However, the distribution of datasets to cloud or edge servers for monitoring and analysis of Android logs has not been sufficiently addressed. This study presents a new approach for conducting security analysis while reducing the impact on system resources and battery consumption. This was achieved by transferring the analysis stage to the edge and utilising the device's collected data. The current research paper is differentiated from prior research in the field of Android security by proposing a unique approach to the security analysis of edge devices, which focuses on the retrieval and distribution of log files for analysis.

The aim of this research was to minimise the impact on system resources and battery consumption by performing the analysis at the edge instead of on the device itself. This approach is supported by the proposed Chidroid, a software package that provides a comprehensive solution for log retrieval and data distribution from Android devices. The integrated toolchain, with Chidroid as the central component, enables the generation, collection, parsing, and distribution of datasets for the development of machine and deep learning models.

### 1.2. Contribution

The current study proposes a unique approach for conducting security analysis on edge devices, which involves the retrieval of logs from Android devices. The aim was to minimise the impact on system resources and battery consumption by transferring the computational workload off the device and executing the analysis at the edge.

Chidroid is a software package that provides a comprehensive solution for log retrieval and data distribution from Android devices. This package leverages a protected policy-driven decomposition technique to enable the generation and collection of logs and thus contribute to the development of detection and analysis tools. This study proposes a modular toolchain, with Chidroid as the central component for generating, collecting, parsing, and distributing datasets. The proposed approach was designed to be reusable and scalable and can be extended to address additional classes of vulnerabilities. Furthermore, Chidroid opens up several possibilities for further applications, such as permission monitoring. The contribution of this research paper and the characteristics of Chidroid are summarised as follows:

- Chidroid is an Android log retrieval and data distribution process that enables the generation and collection of logs from Android devices.
- The evaluation process was conducted in the healthcare domain and interconnected devices such as the IoT and health wearables.
- Chidroid enables the creation of datasets, allowing non-structured data to be parsed into semi-structured or structured data that can be used for machine learning and deep learning. The proposed approach can be used as a universal policy-based tool to identify and analyse security issues in all recent versions of Android (<https://www.appbrain.com/stats/top-android-sdk-versions>, accessed 20 February 2023), approximately 95% of all active Android devices.
- A modular toolchain is proposed, with Chidroid as the main component to generate, collect, and distribute datasets. The approach is reusable, can be extended to detect additional classes of vulnerabilities, and can be applied to permission monitoring.
- Chidroid can retrieve the set of priorities for the applications and can also set the time duration and privileges used by the software packages. The detection rules can report on permissions if an application misbehaves due to the usage or to maintain excessive system resources.

Finally, the research paper provides practical examples of log retrieval from Android devices and details the ways in which these logs can be utilised to extract meaningful insights and behaviour patterns in device usage. A comprehensive examination of the feasibility and potential of this approach is also presented.

### 1.3. Structure

The rest of this paper is structured as follows. Section 2 presents the methodology employed to develop and assess Chidroid and the successive stages followed for this research. Section 3 provides a comprehensive analysis of Chidroid's architecture, including a thorough examination of its constituent components. Section 4 offers practical examples of Chidroid's usage and presents the findings from its evaluation. Finally, the conclusions and potential future avenues for this research are outlined in Section 5.



## 2. Methodology

In this section, the methodology followed during the development of Chidroid is described. The test and validation cycles (Figure 1) were as follows: (i) Chidroid development and versioning updates and the creation of the Android Package Kit (APK) file, (ii) the definition of the default configuration files and permissions granted required by the APK, (iii) enabling Chidroid to execute shell commands defined by the configuration, (iv) the definition of the commands that generate and create Android logs, (v) the creation of data type patterns, regular expressions, and corresponding log sources, and finally, (vi) the support of bash shell piped commands to be executed by Chidroid. These stages were followed sequentially, and multiple iterations were performed to provide better versions of the Chidroid to support the required functionalities. Figure 1 explains the structured methodology to ensure that the application met the functional requirements and performed as expected.



**Figure 1.** Methodology followed to develop, validate, and upgrade Chidroid.

The first step of the methodology (Figure 1) comprises the development of functional requirements, including log retrieval and log distribution. The test and validation steps were carried out to ensure correctness. Once the application was functional, the next step was to define the permissions needed by Chidroid and define the relevant files that should be included in the APK, for example the permissions that Chidroid needs to have to retrieve logs and configuration files that must be included in the APK to enable log collection and shipping by default.

The Chidroid system was derived from a previous iteration of a software application known as Metago, which was developed by the PDMFC organisation. The encoding protocols and formatting techniques utilised in Metago were retained in the development of Chidroid, with additional features incorporated to facilitate the collection and analysis of the log data from Android devices. Consequently, it may include different configuration options employing the encoding scheme characteristic of Metago, or the collected logs may be distributed in the raw format (Metago Raw).

The present study advanced this by incorporating the functionality of executing user-defined shell commands and specifying their usage in the configuration file. The implementation of the shell command execution enabled the collection and tracking of Android logs through the utilisation of relevant tools such as Logcat, Package Manager, and AppOpsManager (<https://developer.android.com/reference/android/app/AppOpsManager>, accessed 20 February 2023). The initial verification of the data collection accuracy was conducted, followed by the parsing of logs into a structured format, the creation of data type patterns, and the definition of regular expressions to extract specific fields. The implementation of shell pipe commands in Chidroid enabled the execution of multiple commands for searching and filtering logs prior to collection and distribution. A methodological validation was carried out at each step of the implementation process, utilising the following devices for testing purposes:

- Android smartphones and tablets were used for the validation steps (Android Version 6 to Version 12).
- Android IoT devices or gateways, such as routers and hubs for smart healthcare and IoMT devices, such as medical sensors and healthcare systems (the exact models are confidential due to licence agreements) (e.g., smart glasses, smartwatches, medical implants).

The individual testing of the devices was carried out to enhance compatibility and verify the functionalities of Chidroid. During the validation tests, no medical or healthcare data were collected. Chidroid collected logs and metrics from a variety of assets and devices to test and validate the functionalities. Validation tests were instrumental in discovering software flaws, bugs, or permission changes and in expanding the functionality of Chidroid and verifying its correctness. In general, the validation tests demonstrated that the application could retrieve logs and system metrics from various assets and devices.

To evaluate the performance impact of Chidroid on the device, a preliminary assessment was carried out to gather data on CPU and RAM utilisation. The assessment involved the usage of the Android Debug Bridge (ADB (<https://developer.android.com/studio/command-line/adb>), accessed 20 February 2023) shell and the execution of specific commands to obtain the relevant metrics. The commands in Listing 1 were used to retrieve the CPU and RAM usage of a specific process with the package name “com.github.lribeiro.metago.chidroid”.

**Listing 1:** Example structure of TOML files.

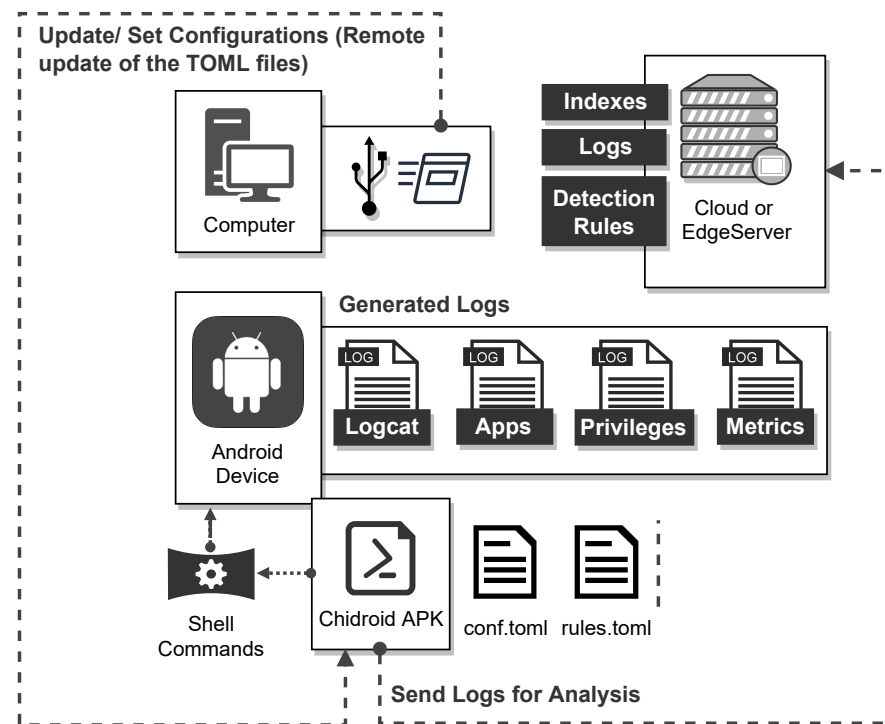
```
1  cpu_usage=$(top -b -n 2 | awk '/com.github.lribeiro.metago.chidroid↵
    / {print $9}')
```

```
2  ram_usage=$(top -b -n 2 | awk '/com.github.lribeiro.metago.chidroid↵
    / {print $10}')
```

The output of the “top” command (Listing 1) provides a comprehensive representation of the processes running on the system. This information includes the process ID (PID), the user who initiated the process, and the CPU and memory utilisation of each process, among other details. In this study, the “top” command was utilised to retrieve the CPU and RAM usage of the Android process with the package name “com.github.lribeiro.metago.chidroid”. The output of the top command was filtered to show only the information relevant to this process. The “awk” command was then utilised to extract the desired metric values, the CPU and RAM usage, from the filtered output. The instant values were stored, and the mean values were calculated by finding the minimum, average, and maximum values.

### 3. Chidroid Architecture

The Chidroid architecture is presented in Figure 2 and describes how the application collects data from Android devices and distributes them to other endpoints. Once the application is installed and configured, it collects logs and system metrics.



**Figure 2.** Chidroid architecture and deployment details.

The Chidroid architecture (Figure 2) allows the efficient collection and secure distribution of logs and system metrics from Android devices. By using shell commands and the API, Chidroid can provide valuable information for monitoring and optimising the performance and security of Android devices. Chidroid generates, collects, and distributes the following logs: (i) logcat.log, (ii) packages.log, and (iii) privileges.log, along with (iv) metrics. Internally, Chidroid uses a scheduler to run user-defined scripts or shell commands to generate the logs or data. This scheduler follows the familiar CRON syntax, extended to add seconds. After collecting logs and system metrics, Chidroid encodes them in JSON and distributes them to the chosen target collector for analysis. This collector can be a cloud service or another web server that is able to receive and process JSON data.

Chidroid can be remotely configured using the POST method, and the API can replace the configuration files, namely config.toml and rules.toml. When Chidroid starts up, it reads these configuration files and sets up the log input sources, internal settings, and output sinks from config.toml. From config.toml, Chidroid reads the processing and transforming rules, for example, regular expressions or event-breaking rules. It is noteworthy that the rules.toml file possesses the capability to integrate privacy-centric rules, which can be invoked during the course of log collection. Such rules serve to enable data transformation, which is geared towards privacy enhancement objectives. This is achieved through the application of anonymisation and other pertinent processes to the collected logs, adding privacy features. As for distribution, Chidroid can ship logs in the raw format or JSON encoded. If needed, traffic can be relayed on another Chidroid listener (peer-to-peer distribution).

### 3.1. Chidroid APK

Chidroid was developed using the Go programming language and compiled as APK (with the package name com.github.lribeiro.metago.chidroid) using Gomobile (<https://pkg.go.dev/golang.org/x/mobile/cmd/gomobile>, accessed 20 February 2023). The build process takes the source code and generates multiple binary files for a combination of operating systems (Linux, Windows, MacOS, and Android) and CPU instruction sets (x86\_64, arm64, arm7). Chidroid currently supports the following CPU architectures: (i) Advanced RISC Machine (CPU architecture ARM and ARM64) and (ii) Intel Atom



(x86\_64). The APK file includes resources, assets, the manifest file, and the linked SO library (which contains the majority of the executable code). In the context of software development, manifest files are used to specify the requirements, dependencies, privileges, and other details that are needed to run the project or application. In Android application development, manifest files are an essential part of APK files. They are used by the Android operating system to determine how to run an application and the permissions that the application acquires from the Android system.

### 3.2. Configuration Files (Conf.Toml, Rules.Toml, Certs, and Manifest Files)

There are two main Chidroid configuration files included in the APK: (i) “config.toml” and (ii) “rules.toml”. The first (config.toml) is used to set up sources and sinks (outputs), while the second (rules.toml) is used to define the privacy rules and to set up the needed transformations to parse the sources to the required format. Tom’s Obvious, Minimal Language (TOML (<https://toml.io/en/v1.0.0>), accessed 20 February 2023) is a file format used to represent configuration data in a human-readable and easily editable format.

TOML files are commonly used in software development to store configuration data for applications and other tools. The files use a simple and consistent syntax, with key–value pairs separated by an equal sign (=) and sections indicated by square brackets ([ and ]). A generic example is provided in Listing 2 and contains two stanzas (i.e., block of text within a configuration file). In this example, the server section includes a single option, “port”, which specifies the port number on which the server will listen for incoming connections.

**Listing 2:** Example structure of TOML files.

---

```

1 [server]
2 port = 8080
3
4 [database]
5 host = "localhost"
6 user = "admin"
7 password = "secret"

```

---

### 3.3. Configuration of Endpoints

A web API endpoint is a specific URL or location on a web server that provides access to a web API. Chidroid supports the distribution of raw data to remote HTTP or HTTPS endpoints. Alternatively, another Chidroid application can act as a gateway with a dedicated listener and receive and forward whatever data it receives to the specified final endpoint. Next, an example of configuring a remote output is presented (Listing 3).

**Listing 3:** Configuration of the outputs.

---

```

1 [outputs.AI4HEALTHSEC]
2 Urls = ["METAGO://siem.AI4HEALTHSEC.pdmfc.com/dynamic-ports←
        /10000/:80/"]

```

---

In Listing 3 a new output named “AI4HEALTHSEC” is defined and specifies that it will send data to a single endpoint using an HTTP Post over HTTPS, to a host where an SIEM is deployed. As an example, in Listing 4, the encoded JSON output is presented as an event retrieved from Logcat (see Section 3.4 for more details on Logcat).

**Listing 4:** Chidroid: METAGO-encoded JSON output.

---

```

1 {
2   "header": {
3     "agentId": "55bb84e8-b2ee-4af4-94f3-435826e14f98",
4     "host": "Android",
5     "os": "Android",
6     "architecture": "arm64",

```

```

7  "source": "script://Logcat"
8  "sourcetype": "logcat"
9  "format": "METAGO"
10 "timefield": "timestamp",
11 "timeformat": "MM-DD hh:mm:ss.SSS"
12 "timezone": "Europe/Lisbon"
13 },
14 "events": [
15 {
16   "timestamp": "12-05 17:48:28.215",
17   "pid": 26458,
18   "opid": 26458,
19   "type": "V",
20   "AppName": "GraphicsEnvironment",
21   "Message": "ANGLE Developer option for 'com.google.android.apps.↵
        messaging' set to: 'default'"
22 }
23 ]
24 }

```

### 3.4. Logcat and Generation of System Logs in the Android Environment

Logcat is a software tool that can retrieve logs available on Android devices. Android stores the kernel logs generated by the system and application logs generated by individual applications. In general, logs are an important source of information for detecting and troubleshooting problems on Android devices. By regularly checking the log data, security professionals can identify potential security threats and take the appropriate action to address them. Even if the main goal of Logcat is to be used as an essential debugging tool, the continuous stream of system messages and user-generated diagnostics is used to detect security issues.

Traditionally, to retrieve logs from an Android device, developers or advanced users connect their computers to the device using the ADB. The simpler way is to connect using USB, install the required drivers if necessary, and open a shell using the *adb shell* command. Inside that shell, the command *logcat* can be issued to retrieve the logs stored internally. The following taxonomy of events is retrievable by Logcat (Table 2).

The logs from Logcat contain information about the device's hardware and software, as well as any message that has been logged by the system or applications, such as the device's boot process, system events, and running applications. Logcat can be combined with arguments or other shell commands to search log data for specific keywords or phrases (e.g., "*adb logcat \*:D*" or using the *grep* command). The logs from Logcat are, by default, stored as circular memory buffers on the device. Therefore, Chidroid can execute and collect the logs from Logcat and distribute the data for analysis. They are collected and shipped before being naturally erased by filling in all the reserved ring buffer sizes.

Several commands can be used on an Android device to generate logs and perform various actions. These commands are typically executed using a command-line interface, such as a terminal emulator application or the ADB tool. Chidroid uses an internal interpreter to execute shell commands that can be scheduled with CRON-like rules. The shell commands to execute Logcat are defined in the *config.toml* file (Listing 5).

**Table 2.** Logcat event types.

Event Type	Description
Log.Verbose	Verbose logs are detailed logs that provide a comprehensive record of an activity or system. In the context of Android, verbose logs may be used to troubleshoot issues with the operating system or a specific app.
Log.Debug	Debug logs are not intended for production environments, as they can contain sensitive information and may impact an app's performance. When an application is ready for release, debug logs should be removed or disabled to ensure that the application runs efficiently and securely.
Log.Info	Info refers to log messages that provide useful information about the state or behaviour of an application or the Android system.
Log.Warning	Warning log messages are typically used to alert the user or the developer to a potential problem or issue that may require attention. For example, a warning log message may be generated if an application receives invalid input from the user, if a network connection is lost, or if a critical system resource is running low.
Log.Error	Error log messages are typically used to alert the user or the developer to a serious problem or failure that has occurred within an application or the Android system. For example, an error log message may be generated if an application crashes or an important system function fails.
Log.Fatal	Fatal log messages should be used sparingly, as they indicate a critical failure or an error that requires immediate attention. When a fatal log message is printed, the application or system may be unable to continue running and may need to be restarted or shut down.

**Listing 5:** Android shell commands defined in the TOML configuration file.

```

1 [script.logcat]
2 Cmd = "logcat -t '#timestamp' -b all"
3 EvalArgsIndex = [1]
4 Outputs = ["AI4HEALTHSEC"]
5 Rules=["eventbreak.eventperm", "rex.logcat"]
6 Index= "android_logs"
7 SourceType="logcatraw"
8 Debug=true
9 Cron="!*/15 * * * * *"
10 [script.logcat.Params]
11 timestamp='01-01 00:00:00.000'

```

There is much unpack here, and the snippet above (Listing 5) corresponds to the execution and log collection using *[script.logcat]*. The “Cmd” property specifies the command that will be executed to collect data. Here, the *logcat* command is executed with the “-t” and “-b” options to filter the output by timestamp and log buffer, respectively. Proper shell-argument parsing is performed on “Cmd”, and all arguments are stored internally in a string array. The “EvalArgsIndex” property specifies the indexes of the script arguments that shall be evaluated as internal expressions. In this case, all but *#timestamp* are simple strings. The *#timestamp* represents the extracted timestamp field of the *rex.logcat* rule presented below.

“Outputs” specifies the names of the output streams that will receive the script’s output. The “Rules” property specifies the names of the rules that will be applied to the script’s output. The “Index” specifies the name of the index that will be used to store the processed log data. The “SourceType” option specifies the type of log data the script collects. The “Debug” option specifies whether debug information will be output to the internal log of Chidroid. The “Cron” option specifies the schedule and recurrence rule for executing the script. The “[script.logcat.Params]” section of the configuration file contains additional parameters passed to the script at runtime. In the example provided above, the “timestamp” parameter was set to “01-01 00:00:00.000”, which for all intents and purposes sets the initial value for the timestamp.

### 3.5. AppOpsManager

AppOps includes a set of APIs within the Android operating system to manage the permissions granted to applications. These APIs effectively monitor and control software packages' permissions on an Android device, enabling the detection and prevention of potential security issues. The proposed methodology entails the utilisation of the "appops" command in the Chidroid software package to access the AppOps interface and retrieve the permissions assigned to each software package. By comprehensively analysing these permissions, Chidroid can identify any deviation or anomaly that may pose a security risk. For instance, if a software package is found to be requesting an excessive number of permissions or utilising permissions that are not congruent with its intended function, Chidroid can alert the user or administrator and take necessary measures to resolve the issue. This feature of Chidroid demonstrates the potential of utilising AppOps APIs to enhance the security of Android devices and safeguard against potential security threats.

Healthcare organisations must comply with the regulations set forth by the Health Insurance Portability and Accountability Act (HIPAA) when handling medical or healthcare data on Android devices. The use of AppOps can assist in enforcing these policies and regulations by tracking and managing the permissions of software packages that handle sensitive data. Chidroid can ensure that these packages are only granted the necessary permissions for their intended use and prevent the use of any permissions that could compromise the security of the device or its users. Using AppOps, Chidroid can monitor the permissions of software packages on Android devices and detect and prevent any security issues that may arise from their use. The privilege.log file generated by AppOps can also assist Chidroid in maintaining the security and HIPAA compliance of medical or healthcare data on Android devices in the eHealth environment.

### 3.6. Android Permission Retrieval

Overall, the config.toml file defines the behaviour and settings of the data collection inputs. Here, the shell commands that initiate the Package Manager (pm command) and the corresponding permission monitoring and retrieval are presented. The "pm" command (Listing 6) is a tool provided by the Android operating system for managing installed packages.

The Package Manager is an Android system internal application that performs actions and queries on applications and software packages installed on an Android device. The command pm is a Command-Line Interface (CLI) API to interact with Package Manager to enumerate or manage the installed packages on the device. It can also enable various actions on those packages. For example, the command can retrieve the full list of the software packages installed on the device, displaying the names of the software packages, along with the version number, process id, and other details for each of the installed software packages.

Internal filtering allows this command to display specific packages, such as associated files, enabled/disabled packages, system packages, third-party packages, and installer packages. For example, the -u flag can display a list of uninstalled packages. In summary, the command can be used to retrieve the lifecycle of installed software packages.

**Listing 6:** Command to retrieve the permissions given to the installed packages.

```

1 [script.permissions]
2 Cmd="pm list packages -u -3
3   | cut -c 9-
4   | while read line; do echo $line; appops get $line
5   | sed 's/Uid mode: //' ; done"
6 Rules=["eventbreak.eventperm", "rex.extract", "zip.perms", "str.j13", "↵
   for.join13", "zip.perms_time", "str.j15", "for.join15", "zip.↵
   perms_duration", "fields.remove_temp"]
7 Index="android_permissions"
8 Outputs = ["AI4HEALTHSEC"]

```

The “list packages” subcommand lists all installed packages on the device, including both user-installed and system packages. The output will include uninstalled packages using the “-u” option, and the “-3” option will include third-party packages. The “pm list packages” output is passed to the “cut” command, which extracts the package names from the output. Using a “while” loop, the script reads the package names one by one and runs the “appops” command to retrieve application permission data for each package. The “appops” command is another tool provided by Android for managing application permissions, and the “get” sub-command retrieves the application permission data for a specific package. The output from the “appops get” command is piped to the “sed” command, which removes the “Uid mode:” prefix from each line of output.

### 3.7. System Metrics

In addition to the aforementioned logs, Chidroid can directly retrieve system metrics from the Android device, including information about battery life, virtual memory usage, CPU usage, and network connections. These system metrics are important to monitor the performance and overall state of the device. Tracking these metrics makes it possible to identify software packages that consume a large number of system resources, such as CPU or memory. This is particularly important in IoMT and healthcare services, where the high usage of resources can lead to risky and critical safety conditions for patients. As a result, Chidroid can help identify and mitigate the potential security risks posed by software packages that consume a large number of system resources. This can help ensure the safety and security of patients and their data in the IoMT and healthcare domains if there is a malfunction, critical system, or security issue.

Chidroid retrieves various metrics, including CPU usage, virtual memory, disk counters, load average, network counters, connections, and running processes, on an Android device using the following stanza (Listing 7). Within this stanza, the index name, job frequency, and endpoint for log distribution are defined for each metric. The source-type settings are described in the following subsections (Sections 3.3 and 3.8).

#### Listing 7: Retrieve device metrics.

---

```

1 [metrics.all]
2 Metrics = [
3   "cpu", # Cpu time consumed
4   "cpuinfo", # Cpu version, and spec
5   "virtualmem", # Memory used/free
6   "disk", # Disk IO counters
7   "loadavg", # CPU load average 1,5,15m
8   "net", # Net IO Counters by interface
9   "connections",
10  "process",
11 ]
12 Cron="!0 * * * * *"
13 Index="android_metrics"
14 Outputs = [ "AI4HEALTHSEC" ]

```

---

The stanza “[metrics.all]” specifies the settings for the “metrics” rule type and creates a new internal instance associated with the name “all”. The “Metrics” property specifies the list of counters to be collected. In the example above (Listing 7), the rule set will collect metrics for CPU time consumed, CPU version and specification, memory usage, disk IO counters, CPU load average, network IO counters, network connections, and processes.

The “Cron” property specifies the schedule at which the metrics will be collected. Metrics will be collected every minute (0 \* \* \* \*). The “Index” suggests the target database name where metrics should be stored. Log collectors have final control and can override this definition. The “Outputs” option specifies the next hop the collected metrics will be



sent. Here, the metrics will be sent to the “AI4HEALTHSEC” as the configuration output (see Section 3, Listing 3).

### 3.8. Log Distribution for Analysis—Regex and Log Sourcetypes

Regular expressions are a powerful technique for matching and manipulating text and are widely used in various fields of study and industry. Regular expressions can be employed in log sources to search and filter data for specific patterns or messages. A regular expression is a string of characters that defines a pattern to be matched in a text string. These patterns are specified using a special syntax, which allows for the matching of a wide range of patterns in text. For example, regular expressions can be used to match a specific word or phrase in a log message or to identify a specific pattern of characters in a URL. To search for a specific pattern in a log message using a regular expression, the pattern can be defined using the regular expression syntax and then employed with a tool or program such as Chidroid, which has built-in support for regular expressions. The tool or program will highlight or indicate the matching text in the log data if the pattern is found. Chidroid also has the capability of utilising named captures, a feature of regular expressions, which allows for the capturing of text matched by the regular expression and the ability to reuse it within the same pattern or in a substitution. This feature allows for greater flexibility and precision in analysing and manipulating log data.

Each capture group in the example below (Listing 8) is defined using the following syntax: “(?P<name>pattern)”, where “name” is the name of the capture group and “pattern” is the regex pattern that defines the contents of the group. The regex pattern in the example has five named capturing groups: “timestamp”, “pid”, “opid”, “type”, “AppName”, and “Message”. The “timestamp” capturing group matches a timestamp in the format “MM-DD hh:mm:ss.SSS”. The “pid” and “opid” capturing groups match a string of digits representing a process ID and operation ID, respectively. The “type” capture group matches a single word character representing the type of the log message. The “AppName” capture group matches a string of characters representing the name of the application that generated the log message up to the first colon character. The “Message” capture group matches the remaining characters in the log message after the application name.

#### Listing 8: Configuration of the regular expressions and source types.

```
1 [rex.logcat]
2 Patterns=[ '(?P<timestamp>\d\d-\d\d \d\d:\d\d:\d\d.\d\d\d)\s+(?P<pid←
   >\d+)\s+(?P<opid>\d+)\s+(?P<type>\w)\s+(?P<AppName>[^:]+):\s+(?P←
   <Message>.*) ' ]
3 Field="_raw"
4
5 [sourcetypes.logcat] #Define the Time Format of the logs
6 Format="METAGO"
7 TimeFormat="MM-DD hh:mm:ss.SSS"
8 TimeField="timestamp"
9 TimeZone="Europe/Lisbon"
```

The “[sourcetypes.logcat]” configuration stanza (Listing 8) creates a new SourceType named “logcat”. The SourceType contains a set of properties that hint at how logs should be parsed, of particular importance for detecting and extracting correct timestamps for each entry. The timezone, timestamp format, and other configuration details are also defined.

### 3.9. Performance Assessment and Compatibility

Chidroid allows the usage of multiple pipelines and sends data in multiple outputs with one execution. Chidroid is compatible with Android smartphones running Version 8.0 or later and has been designed to have minimal impact on system resources. It was assessed through empirical analysis that the system under consideration exhibits a memory footprint of approximately 62 MB in terms of RAM consumption while displaying low CPU

utilisation of approximately 1% on an Octa-core architecture comprised of  $1 \times 2.84$  GHz Cortex-X1,  $3 \times 2.42$  GHz Cortex-A78, and  $4 \times 1.80$  GHz Cortex-A55 processors. The values in the outputs (Listing 9) represent the mean CPU and RAM usage of the Android process. The date and time of the results of the assessment are included in each output, along with the mean CPU usage and mean RAM usage in percentage. These metrics give information on the resource utilisation of the process at a specific moment.

**Listing 9:** Example structure of TOML files.

---

```

1 #Output from the lower CPU usage during the assessment
2 { "date": "2023-02-11_18-59-45", "mean_cpu_usage": 3%, "↵
   mean_ram_usage": 1.8% }
3 #Output from the mean CPU usage during the assessment
4 { "date": "2023-02-11_19-20-01", "mean_cpu_usage": 6.2%, "↵
   mean_ram_usage": 1.6% }
5 #Output from the higher CPU usage during the assessment
6 { "date": "2023-02-11_18-48-44", "mean_cpu_usage": 10.3%, "↵
   mean_ram_usage": 1.8% }

```

---

#### 4. Log Generation and Analysis: Examples, Results, and Discussion

In this section, a collection of sample log files is presented that was retrieved from various Android devices. These logs serve as exemplars of the Chidroid execution process and demonstrate the data types that can be obtained through this methodology. The log files can be extracted from Android devices and transmitted to the edge server for further analysis. Through the utilisation of regular expression matching, the data contained within the logs can be transformed into a structured format, enabling the ease of monitoring. Additionally, the data can be exported in a variety of formats, including JSON and CSV, for further analysis and manipulation.

##### 4.1. Logcat Example

The following snippet presents an example output using Logcat. These logs contain different events, including system events, application events, and debugging messages. They can be useful for detecting security issues such as unauthorised access attempts or abnormal application behaviour. The default Logcat structure is explained below (Listing 10).

**Listing 10:** Explanation of the Logcat default structure.

---

```

1 [08-01-2022 10:15:23] [DEBUG] [MyApp] [MainActivity]: Starting activity: Intent { cmp=com.↵
   example.myapp/.MainActivity }
2 [08-01-2022 10:15:24] [INFO] [MyApp] [MainActivity]: Successfully connected to database
3 [08-01-2022 10:15:29] [WARNING] [MyApp] [MainActivity]: Invalid input detected
4 [08-01-2022 10:15:32] [ERROR] [MyApp] [MainActivity]: Failed to retrieve data from server

```

---

In the sample Logcat output, there are four log messages (DEBUG, INFO, WARNING, ERROR), each with a different log level. The first log message is a debug message indicating that the application is starting an activity. The second log message is an info log message, indicating that the application has successfully connected to a database. The third log message is a warning message indicating that the application has received invalid user input. The fourth log message is an error log message indicating that the application has failed to retrieve data from a server. In the Logcat output (Listings 10 and 11), each log message is printed on a separate line and includes the following information:

- Timestamp: The date and time at which the log message was generated, in the format YYYY-MM-DD HH:MM:SS.
- Log level: The severity of the log message, which can be one of the following: DEBUG, INFO, WARNING, ERROR, or FATAL.
- Application name: The name of the application or system that generated the log message.

- Log tag: A short string that provides additional context about the log message.
- Log message: The actual log message that provides information about the state or behaviour of the application or system.

**Listing 11:** Example output using Logcat.

---

```

1 12-05 17:48:28.215 26458 26458 V GraphicsEnvironment: ANGLE Developer option for 'com.google↵
  .android.apps.messaging' set to: 'default'
2 12-05 17:47:04.950 8113 8113 D IHansCommunication: HansCommunication::configCheckedUid-- ↵
  10302, type = 0
3 12-05 17:46:52.935 2058 3726 I OplusHansManager : uid=10302, pkg=com.spotify.music F exit() ,↵
  reason=Broadcast
4 12-05 17:46:52.935 3801 3801 D DeviceInfoHidlClient: isRadioOn()=true
5 12-05 17:48:28.517 2058 7258 W PackageManager: package unknown uid 10223 pid 26458 call ↵
  SetEnabledSetting(com.google.android.apps.messaging component = com.google.android.ims.↵
  binding.SystemBindingService, 1, 0x1, 0)
6 12-05 17:48:28.519 26458 26492 I BugleRcsEngine: [1364] bdre.run: SystemBindingManager: ↵
  SystemBinding enabled: true
7 12-05 17:48:28.520 26458 26492 I BugleRcsEngine: [1364] bdrf.b: System Binding updated
8 12-05 17:47:04.022 3872 7096 E Battery : AppStats: Uid = 10317; pkgName = gr.winbank.↵
  mobilenext; GpsPower = 1.253625 mAh; GpsTotalPower = 0.0 mAh

```

---

By regularly checking logs (Listing 11), it is possible to identify potential security risks and take the appropriate actions to mitigate them. For example, if the log contains a message like “User logged in successfully”, this indicates that a user has successfully logged into the app. However, if the log contains a message like “User login failed”, this may indicate a potential security risk, such as an incorrect password or username used in an attempted login. In this case, it would be important to investigate the issue further and take appropriate action to secure the system, such as requiring the user to reset his/her password or blocking the device’s IP address that made the failed login attempt.

#### 4.2. Package Manager

The Package Manager is a system service on Android devices and is used to manage the installation, removal, and updating of software packages. One of the tasks that can be performed using the Package Manager is to enumerate the list of software packages installed on the device. To do this, the following command can be used (Listing 12):

**Listing 12:** Enumerate installed packages.

---

```

1 pm list packages

```

---

This command (Listing 12) will display a list of all software packages installed on the device, along with their package names and versions (Listing 13). For example, the output of the command “pm list packages” might look as follows.

**Listing 13:** Installed software packages.

---

```

1 com.skype.raider:8.66.0.123
2 cn.wps.xiaomi.abroad.lite:6.2.2
3 com.aviapp.translator:1.3.5
4 com.adobe.reader:20.6.0
5 com.linkedin.android:16.0.0.226
6 com.termux:0.115

```

---

In this example, the package names are listed on the left, followed by the version numbers in parentheses. The version number consists of three or four parts, separated by periods. The first number indicates the major version, the second the minor version, and the third the patch level. The fourth number, if present, indicates a build number.

The command “pm list packages” can be used to obtain a list of all installed packages on the device, or the “-f” flag can filter the output based on a specific package name or part of a package name. For example, to list all installed packages that contain the word “skype” in the package name, use the following command (Listing 14).

**Listing 14:** Filter by package name.

---

```
1 pm list packages -f "skype"
```

---

To retrieve detailed information about a package, you can use the following command (Listing 15).

**Listing 15:** Retrieve package information.

---

```
1 pm dump <package_name>
```

---

The “<package\_name>” argument should be replaced with the application’s package name to obtain the relevant information. The package name is a unique identifier for each application that the Android operating system uses to identify and manage the application. The “pm dump” command will display detailed information about the specified package, including its permissions, version, and signature. This information can be helpful when debugging issues with the application or when trying to understand its behaviour. Here are some examples of package names and the applications to which they correspond:

**com.skype.raider** This is the package name for the Skype (<https://www.skype.com/>, accessed 20 February 2023) application, which is a messaging and video calling application developed by Skype.

**cn.wps.xiaomi.abroad.lite** This is the package name for the WPS Office (<https://www.wps.com/>, accessed 20 February 2023) application, which is a productivity suite developed by Kingsoft Office Software Corporation.

**com.adobe.reader** This is the package name for the Adobe Acrobat Reader (<https://get.adobe.com/reader/>), a PDF reader and annotator developed by Adobe Systems.

**com.linkedin.android** This is the package name for LinkedIn (<https://www.linkedin.com/>, accessed 20 February 2023), which is a professional networking and job searching application developed by LinkedIn Corporation.

**com.termux** This is the package name for Termux (<https://github.com/termux/>, accessed 20 February 2023), which is a terminal emulator and Linux environment for Android.

The package names can provide a way to identify and manage the applications installed on an Android device.

#### 4.3. AppOpsManager and Privilege Extraction

The following stanza (Listing 16) includes sample logs retrieved from an Android smartphone using the Package Manager to enumerate the installed software packages. These logs display the package names, version numbers, and permissions of the packages installed on the device. Package names are unique identifiers assigned to each application by the developer, while version numbers indicate the current version of the application installed on the device. The permissions indicate the actions and data the application can access on the device.

**Listing 16:** Example output from the permission parser.

---

```
1 com.linkedin.android
2 Uid mode: COARSE_LOCATION: ignore
3 FINE_LOCATION: ignore
4 READ_CONTACTS: ignore
5 WRITE_CONTACTS: ignore
6 READ_CALENDAR: ignore
```

---

```

7 CAMERA: ignore
8 RECORD_AUDIO: ignore
9 READ_PHONE_STATE: ignore
10 SYSTEM_ALERT_WINDOW: default; rejectTime=+2d18h29m18s525ms ago
11 READ_CLIPBOARD: allow; time=+71d4h26m24s53ms ago
12 WAKE_LOCK: allow; time=+10h57m51s192ms ago; duration=+265ms
13 USE_BIOMETRIC: allow; time=+1h41m59s218ms ago

```

---

The logs show the permissions of LinkedIn (<https://www.linkedin.com/>, accessed 20 February 2023), namely “com.linkedin.android”. By tracking the permissions of these packages, it is possible to detect any abnormal usage of permissions, such as a game requesting permission to write to external storage when it should not. This can help ensure that the software packages are not violating any regulations or policies and do not present any security risks to the device or its users. Another example (Listing 17) follows for the Termux application.

**Listing 17:** Permissions on the Termux package.

---

```

1 com.termux
2 Uid mode: LEGACY_STORAGE: allow
3 ACCESS_MEDIA_LOCATION: ignore
4 SYSTEM_ALERT_WINDOW: ignore
5 READ_CLIPBOARD: allow; time=+38d16h15m20s598ms ago
6 WRITE_CLIPBOARD: allow; time=+38d16h21m13s703ms ago
7 START_FOREGROUND: allow; time=+20d16h15m12s687ms ago; duration=+40←
    s7ms
8 MANAGE_EXTERNAL_STORAGE: default; rejectTime=+64d2h40m4s128ms ago

```

---

The above logs (Listing 17) present the permissions requested by the Termux application for Android. The application has been granted the “LEGACY\_STORAGE”, “READ\_CLIPBOARD”, “WRITE\_CLIPBOARD”, and “START\_FOREGROUND” permissions by the user, but has not been granted the “ACCESS\_MEDIA\_LOCATION” or “MANAGE\_EXTERNAL\_STORAGE” permission. The application requests the following permissions:

- LEGACY\_STORAGE: This permission allows the application to access the device’s legacy storage directories, which may be used to store data compatible with older versions of Android.
- ACCESS\_MEDIA\_LOCATION: This permission allows the application to access the location metadata associated with media files on the device, such as photos and videos.
- SYSTEM\_ALERT\_WINDOW: This permission allows the application to display windows on top of other applications, which may be used to display important information or to provide additional functionality.
- READ\_CLIPBOARD: This permission allows the application to read from the device’s clipboard, which may be used to paste text or other data into the app.
- WRITE\_CLIPBOARD: This permission allows the application to write to the device’s clipboard, which may be used to copy text or other data from the app.
- START\_FOREGROUND: This permission allows the application to start a foreground service, which will continue to run even if the application is not in the foreground.
- MANAGE\_EXTERNAL\_STORAGE: This permission allows the application to manage the device’s external storage, such as an SD card, which may be used to store application data or other files.

The temporal metrics “time” and “duration” for the permissions “READ\_CLIPBOARD”, “WRITE\_CLIPBOARD”, and “START\_FOREGROUND” serve to record the point in time when the user granted the permission, as well as the duration for which the application has utilised the permission. As an illustration, the “time” value for the “READ\_CLIPBOARD” permission reveals that the user authorisation was granted 38 days, 16 h, 15 min, 20 s, and 598 ms prior to the



current time. Conversely, the “duration” value for the “START\_FOREGROUND” permission demonstrates that the application has been in possession of the permission for a period of 40 s and 7 ms. In conclusion, these values furnish insight into the manner in which applications obtain the permissions granted by the user.

The “Uid mode” values for each permission indicate how the application will handle the permission. For example, the “ignore” value for the “ACCESS\_MEDIA\_LOCATION” permission indicates that the application will not use the permission, even if it is granted by the user. The “default” value for the “MANAGE\_EXTERNAL\_STORAGE” permission indicates that the application will use the default system behaviour for permission, which may be to ask the user for permission when the application attempts to access the protected data or resource.

#### 4.4. System Metrics

System or device metrics (Listing 18) are essential for monitoring the performance and general status of an Android device. These metrics can provide valuable information about various aspects of the device’s hardware, software, and network configuration, which can be useful for troubleshooting issues, identifying trends, and optimising performance.

**Listing 18:** Example output of the system metrics retrieved from Android.

```

1 2022-11-28 15:19:00 @date :11/28/2022 CoreID :0Cores :1Flags :fp asimd evtstrm aes pmull ↵
    sha1 sha2 crc32Mhz :1872Model :0xd03Stepping :4VendorID :ARMmetric :cpuinfo
2 2022-11-28 15:19:00 @date :11/28/2022 CPU :1CoreID :1Cores :1Flags :fp asimd evtstrm aes ↵
    pmull sha1 sha2 crc32Mhz :1872Model :0xd03Stepping :4VendorID :ARMmetric :cpuinfo
3 2022-11-28 15:19:00 @date :11/28/2022 CPU :2CoreID :2Cores :1Flags :fp asimd evtstrm aes ↵
    pmull sha1 sha2 crc32Mhz :1872Model :0xd03Stepping :4VendorID :ARMmetric :cpuinfo
4 2022-11-28 15:19:00 @date :11/28/2022 CPU :3CoreID :3Cores :1Flags :fp asimd evtstrm aes ↵
    pmull sha1 sha2 crc32Mhz :1872Model :0xd03Stepping :4VendorID :ARMmetric :cpuinfo
5 2022-11-28 15:19:00 @date :11/28/2022 available :1526226944free :133947392metric :↵
    virtualmemtotal :2952015872used :1327394816
6 2022-11-28 15:19:00 @date :11/28/2022 load1 :4.78955078125load15 :5.0224609375load5 :4.78125↵
    metric :loadavg
7 2022-11-28 15:19:00 @date :11/28/2022 bytesRecv :288737228bytesSent :11125292metric :netname↵
    :wlan0packetsRecv :234757packetsSent :81621
8 2022-11-28 15:19:00 @date :11/28/2022 bytesRecv :176bytesSent :176metric :netname :↵
    lopacketsRecv :2packetsSent :2
9 2022-11-28 15:19:00 @date :11/28/2022 metric :netname :rmnet2
10 2022-11-28 15:19:00 @date :11/28/2022 dst_ip :213.63.130.243dst_port :443family :2fd :75↵
    metric :connectionspid :21859src_ip :192.168.1.88src_port :48582status :SYN_SENTtype :1

```

The log messages (Listing 18) provide information about the device’s CPU, including the core ID, the number of cores, the CPU flags, the CPU speed, and the CPU vendor ID (Lines 1 to 5 in Listing 18). Other log messages provide information about the device’s memory usage, load average (Line 6), network traffic (Line 7), and network connections (Line 10). For example, one of the log messages indicates that the device has four CPU cores, each with a speed of 1872 MHz. The log message also indicates that the CPU supports various instruction sets, and details regarding the CPU (ARMv8-A) are provided. Other log messages provide information about the device’s memory usage, including the total amount of memory, free memory, and used memory. The log message indicates that the device has 2.952.015.872 bytes of memory, with 1.327.394.816 bytes currently in use.

Finally, the log messages provide information about the device’s network connections, including the source and destination IP addresses, the port numbers, the protocol family, and the connection status. For example, one of the log messages indicates that the device has established a network connection to the IP address 213.63.130.243 on port 443 using the TCP protocol. The connection is currently in the “SYN\_SENT” state, which indicates that the device has sent a synchronisation request, but has not yet received a response.

Each log message includes a timestamp indicating the date and time the message was generated. The messages include fields such as the CPU core ID, the number of CPU cores, the CPU flags, the CPU speed, and the CPU vendor ID. Other log messages provide information about device memory usage, load average, network traffic, and network connections:

- CPU usage: This metric indicates the percentage of CPU currently being used by the device. High CPU usage can indicate that the device is under heavy load and may be experiencing performance issues.
- Memory usage: This metric indicates the amount of memory the device uses. High memory usage can indicate that the device is running low on memory and may be experiencing performance issues.
- Battery status: This metric indicates the device's current battery level and charging status. A low battery level can indicate that the device may need to be charged more often or consumes a large amount of power.
- Network usage: This metric indicates the amount of data currently being transferred over the network by the device. High network usage can indicate that the device is transmitting or receiving a large amount of data, which can affect performance and battery life.

Overall, these log messages provide a snapshot of the device's performance and configuration, which can be useful for troubleshooting issues and monitoring the device's health. By tracking and analysing these metrics, it is possible to monitor the performance and overall status of an Android device. For example, if the device has high CPU usage and a low battery level, it may be experiencing performance issues and may need to be charged. Similarly, if memory usage is high, it could indicate that the device is running low on memory or that a specific software package consumes a large amount of memory. Regularly checking system metrics makes it possible to identify and address potential performance and security issues on the device. Monitoring the system metrics makes it possible to identify and address potential problems and ensure that the device runs smoothly and securely.

## 5. Conclusions

This research paper presented Chidroid, an innovative mobile Android application for collecting and distributing logs from smart healthcare and IoMT devices. With this cutting-edge tool, healthcare professionals and IT administrators can easily gather critical log data from a wide range of medical devices, including wearable fitness trackers and remote monitoring systems. The intuitive interface allows users to view, filter, and analyse log data in real-time, enabling them to quickly identify and address any potential security issues or performance problems. Additionally, the application's compatibility for distributing data to cloud-based architectures allows for the easy distribution of logs to multiple stakeholders, ensuring that everyone has access to vital information. This way, it is possible to transform how healthcare and IoMT devices are managed. It is important to note that future applications that utilise the collected log data may necessitate the sanitisation and anonymisation of the logs to address privacy concerns. Chidroid includes such functionalities, which are not finalised and were not explained in the context of this research paper. Considering that, this research paper assumed that the edge devices involved in log collection and analysis are secured by implementing appropriate security protocols.

Chidroid enables a novel Android security system that can serve as a universal policy-based tool to examine and find security issues in most Android versions by distributing logs for analysis. The logs can contain valuable information about the device and its applications, such as hardware and software information, system events, and running applications. The metrics collected include battery status, network connections, virtual memory, and CPU usage. These metrics are important for monitoring the performance and overall state of the device. Logs and metrics are retrieved from the Android device using the Logcat and Package Manager tools. Logs from Android devices can be useful for

improving the security of Internet of Medical Things (IoMT) devices in a number of ways. For example, logs can be used for security analysis as follows:

- Detect and diagnose security issues, such as unauthorised access or malicious activity on the device.
- Track changes to the device's configuration and identify potential security risks, such as installing unauthorised software or enabling unsafe permissions.
- Monitor device usage and identify unusual or suspicious activity that may indicate a security threat.
- Assist forensic investigations in the event of a security breach by providing a record of events that can help identify the cause of the issue and the steps taken to resolve it.

Overall, logs can be an important resource for improving the security of IoMT devices by providing visibility into the device's activities and helping to detect and mitigate potential security risks. In addition to logs, Chidroid directly retrieves various system metrics from the Android device, including information about battery status, virtual memory usage, and CPU usage. These metrics are important to monitor the performance and overall state of the device. By regularly retrieving and analysing logs and system metrics, Chidroid can help identify and mitigate potential security and privacy risks for devices and the sensitive data they handle. It can also be used to extract datasets from logs and use them to improve device performance and reliability. In general, Chidroid is a powerful and versatile tool that can help ensure the safety and security of Android devices in the smart healthcare and IoMT domains.

#### *Future Work*

There are several directions in which the Chidroid tool could be improved and extended in the future. Some potential areas of future work include the following:

- Improved log collection and analysis: The Chidroid system aims to enhance its security posture by collecting and analysing logs. Currently, the focus is on securing and protecting the privacy of Android devices. However, further analysis is needed to define detection rules for security incidents. Advanced log analysis techniques such as machine learning algorithms or statistical analysis could potentially provide insight into the performance and security of the device. The security analysis of collected logs represents a promising area for future research.
- Enhanced security and privacy: The Chidroid system is designed to identify and mitigate security and privacy risks in Android devices. However, further analysis is necessary to determine additional ways to improve protection. A key future milestone will be to conduct a security analysis to support the development of more robust detection rules.
- Extended testbed: An eHealth testbed will be established as a platform for conducting experiments and evaluating the performance of the Chidroid system in real-world eHealth applications. The testbed will be flexible and scalable, allowing for the testing of various configurations and parameters.
- Performance assessment: A comprehensive evaluation method will be developed to measure the impact of the Chidroid system in a more comprehensive manner regarding the resource utilisation and performance. This will provide valuable information on the potential of Chidroid as a solution for edge computing applications. Towards this direction, more research will be conducted to examine the feasibility of reducing the resource requirements by scaling down the continuous log retrieval process. The goal is to decrease the CPU utilisation requirements and optimise the resource utilisation of Chidroid.
- Compatibility: While there is potential to extend the Chidroid system to support iOS devices, the current focus will remain solely on Android devices. This is due to the greater opportunity for researching and addressing security and privacy challenges on the open Android platform, as opposed to the closed architecture of iOS.

In general, there are many opportunities to extend and improve the Chidroid tool to make it an even more powerful and effective tool to track the performance, security, and privacy of Android devices in the smart healthcare and IoMT domains.

**Author Contributions:** Conceptualisation, S.K. and L.L.R.; methodology, S.K.; software, L.L.R.; validation, S.K. and L.L.R.; formal analysis, L.L.R.; writing—original draft preparation, S.K.; writing—review and editing, L.L.R., E.M., C.N. and L.M.C.; supervision, E.M. and L.M.C. All authors have read and agreed to the published version of the manuscript.

**Funding:** This work was funded by European Union’s Horizon 2020 Research and Innovation Programmes under Grant Agreement No. 883273 (AI4HEALTHSEC) and No. 101007273 (DAIS).

**Institutional Review Board Statement:** Not applicable.

**Informed Consent Statement:** Not applicable.

**Data Availability Statement:** Sample logs can be obtained by contacting the authors.

**Acknowledgments:** The authors would like to acknowledge the support and funding provided by the AI4HEALTHSEC and DAIS projects. These projects have played a crucial role in the development and execution of the research presented in this paper. We are grateful for the opportunity to collaborate with the esteemed members of these projects and for the resources provided to us throughout the course of our research.

**Conflicts of Interest:** The authors declare no conflict of interest.

## Abbreviations

The following abbreviations are used in this manuscript:

ADB	Android Debug Bridge
ARM architecture	Advanced RISC Machine architecture
API	Application programming interface
AI	Artificial intelligence
APK	Android Package Kit
ARM	Advanced RISC Machine
BMD	Bi-level malware detection
CPU	Control processing unit
CLI	Command0Line Interface
DoS	Denial of service
HIDS	Host intrusion detection system
HIPAA	Health Insurance Portability and Accountability Act
IoMT	Internet of Medical Things
IoT	Internet of Things
MCG	Method call graph
mHealth	mobile Health
PHI	Personal health information
RAM	Random access memory
TOML	Tom’s Obvious Minimal Language

## References

1. Vermesan, O.; Friess, P.; Guillemin, P.; Giaffreda, R.; Grindvoll, H.; Eisenhauer, M.; Serrano, M.; Moessner, K.; Spirito, M.; Blystad, L.; et al. Internet of things beyond the hype: Research, innovation and deployment. In *Building the Hyperconnected Society-Internet of Things Research and Innovation Value Chains, Ecosystems and Markets*; River Publishers: New York, NY, USA, 2022; pp. 15–118. [\[CrossRef\]](#)
2. Almomani, I.; Al Khayer, A. A comprehensive analysis of the android permissions system. *IEEE Access* **2020**, *8*, 216671–216688. [\[CrossRef\]](#)
3. Sarkar, A.; Goyal, A.; Hicks, D.; Sarkar, D.; Hazra, S. Android application development: A brief overview of android platforms and evolution of security systems. In Proceedings of the 2019 Third International Conference on I-SMAC (IoT in Social, Mobile, Analytics and Cloud)(I-SMAC), Palladam, India, 12–14 December 2019; pp. 73–79. [\[CrossRef\]](#)
4. Garg, S.; Baliyan, N. Comparative analysis of Android and iOS from security viewpoint. *Comput. Sci. Rev.* **2021**, *40*, 100372. [\[CrossRef\]](#)

5. He, D.; Naveed, M.; Gunter, C.; Nahrstedt, K. Security concerns in Android mHealth apps. *AMIA Annu. Symp. Proc.* **2014**, *2014*, 645. [PubMed]
6. Jang, J.; Colletti, A.; Ricklefs, C.; Snyder, H.; Kardonsky, K.; Duggan, E.; Umpierrez, G.; O'Reilly-Shah, V. Implementation of App-Based Diabetes Medication Management: Outpatient and Perioperative Clinical Decision Support. *Curr. Diabetes Rep.* **2021**, *21*, 50. [CrossRef] [PubMed]
7. Halouzka, K.; Burita, L.; Kozak, P. Overview of Cyber Threats in Central European Countries. In Proceedings of the 2021 Communication and Information Technologies (KIT), Vysoke Tatry, Slovakia, 13–15 October 2021; pp. 1–6. [CrossRef]
8. Ramsdell, K.; Esbeck, K. MITRE, Health Cyber, EVOLUTION OF RANSOMWARE (2021). Available online: <https://healthcyber.mitre.org/wp-content/uploads/2021/08/Ransomware-Paper-V2.pdf> (accessed on 20 February 2023).
9. Kettani, H.; Cannistra, R. On cyber threats to smart digital environments. In Proceedings of the 2nd International Conference on Smart Digital Environment, Rabat, Morocco, 18–20 October 2018; pp. 183–188. [CrossRef]
10. Bhosale, K.; Nenova, M.; Iliev, G. A study of cyber attacks: In the healthcare sector. In Proceedings of the 2021 Sixth Junior Conference on Lighting (Lighting), Gabrovo, Bulgaria, 23–25 September 2021; pp. 1–6. [CrossRef]
11. Binbusayyis, A.; Alaskar, H.; Vaiyapuri, T.; Dinesh, M. An investigation and comparison of machine learning approaches for intrusion detection in IoMT network. *J. Supercomput.* **2022**, *78*, 17403–17422. [CrossRef] [PubMed]
12. Razdan, S.; Sharma, S. Internet of Medical Things (IoMT): Overview, emerging technologies, and case studies. *IETE Tech. Rev.* **2022**, *39*, 775–788. [CrossRef]
13. Hatzivasilis, G.; Soultatos, O.; Ioannidis, S.; Verikoukis, C.; Demetriou, G.; Tsatsoulis, C. Review of security and privacy for the Internet of Medical Things (IoMT). In Proceedings of the 2019 15th International Conference on Distributed Computing in Sensor Systems (DCOSS), Santorini, Greece, 29–31 May 2019; pp. 457–464. [CrossRef]
14. Vaiyapuri, T.; Binbusayyis, A.; Varadarajan, V. Security, privacy and trust in IoMT enabled smart healthcare system: A systematic review of current and future trends. *Int. J. Adv. Comput. Sci. Appl.* **2021**, *12*, 731–737. [CrossRef]
15. Koutras, D.; Stergiopoulos, G.; Dasaklis, T.; Kotzanikolaou, P.; Glynos, D.; Douligeris, C. Security in IoMT communications: A survey. *Sensors* **2020**, *20*, 4828. [CrossRef]
16. Zhang, Y.; Deng, R.; Zheng, D.; Li, J.; Wu, P.; Cao, J. Efficient and robust certificateless signature for data crowdsensing in cloud-assisted industrial IoT. *IEEE Trans. Ind. Inform.* **2019**, *15*, 5099–5108. [CrossRef]
17. Martinez, A.; Pérez, M.; Ruiz-Martinez, A. A comprehensive review of the state of the art on security and privacy issues in Healthcare. *ACM Comput. Surv.* **2022**. [CrossRef]
18. Otoum, Y.; Chamola, V.; Nayak, A. Federated and Transfer Learning-Empowered Intrusion Detection for IoT Applications. *IEEE Internet Things Mag.* **2022**, *5*, 50–54. [CrossRef]
19. Wright, A.; Aaron, S.; Bates, D. The big phish: Cyberattacks against US healthcare systems. *J. Gen. Intern. Med.* **2016**, *31*, 1115–1118. [CrossRef] [PubMed]
20. Divakaran, J.; Prashanth, S.; Mohammad, G.; Shitharth, D.; Mohanty, S.; Arvind, C.; Srihari, K.; Abdullah, R.Y.; Sundramurthy, V.F. Improved handover authentication in fifth-generation communication networks using fuzzy evolutionary optimisation with nanocore elements in mobile healthcare applications. *J. Healthc. Eng.* **2022**, *2022*, 2500377. [CrossRef] [PubMed]
21. Sihag, V.; Swami, A.; Vardhan, M.; Singh, P. Signature based malicious behavior detection in android. In Proceedings of the International Conference on Computing Science, Communication and Security, Gujarat, India, 26–27 March 2020; pp. 251–262. [CrossRef]
22. Lee, J.; Lee, Y.; Jin, M.; Kim, J.; Hong, J. Analysis of application installation logs on android systems. In Proceedings of the 34th ACM/SIGApplication Symposium on Applied Computing, Limassol, Cyprus, 8–12 April 2019; pp. 2140–2145. [CrossRef]
23. Sasidharan, S.; Thomas, C. ProDroid—An Android malware detection framework based on profile hidden Markov model. *Pervasive Mob. Comput.* **2021**, *72*, 101336. [CrossRef]
24. Wang, H.; Zhang, W.; He, H. You are what the permissions told me! Android malware detection based on hybrid tactics. *J. Inf. Secur. Appl.* **2022**, *66*, 103159. [CrossRef]
25. Zhang, X.; Mathur, A.; Zhao, L.; Rahmat, S.; Niyaz, Q.; Javaid, A.; Yang, X. An early detection of android malware using system calls based machine learning model. In Proceedings of the 17th International Conference on Availability, Reliability and Security, Vienna, Austria, 23–26 August 2022; pp. 1–9. [CrossRef]
26. Jerbi, M.; Dagdia, Z.; Bechikh, S.; Said, L. Android malware detection as a bi-level problem. *Comput. Secur.* **2022**, *121*, 102825. [CrossRef]
27. Ito, K.; Hasegawa, H.; Yamaguchi, Y.; Shimada, H. Detecting privacy information abuse by android apps from API call logs. In Proceedings of the Advances in Information and Computer Security: 13th International Workshop on Security, IWSEC 2018, Sendai, Japan, 3–5 September 2018; pp. 143–157. [CrossRef]
28. Khariwal, K.; Singh, J.; Arora, A. IPDroid: Android malware detection using intents and permissions. In Proceedings of the 2020 Fourth World Conference on Smart Trends in Systems, Security and Sustainability (WorldS4), London, UK, 27–28 July 2020; pp. 197–202. [CrossRef]
29. Ahmad, M.; Costamagna, V.; Crispo, B.; Bergadano, F.; Zhauniarovich, Y. StaDART: Addressing the problem of dynamic code updates in the security analysis of android applications. *J. Syst. Softw.* **2020**, *159*, 110386. [CrossRef]
30. Arzt, S.; Rasthofer, S.; Fritz, C.; Bodden, E.; Bartel, A.; Klein, J.; Le Traon, Y.; Octeau, D.; McDaniel, P. Flowdroid: Precise context, flow, field, object-sensitive and lifecycle-aware taint analysis for android apps. *ACM Sigplan Not.* **2014**, *49*, 259–269. [CrossRef]



31. Androguard: Reverse Engineering, Malware and Goodware Analysis of Android Applications. Available online: <https://code.google.com/p/androguard/> (accessed on 20 February 2023).
32. Wei, F.; Roy, S.; Ou, X. Amandroid: A precise and general inter-component data flow analysis framework for security vetting of android apps. *ACM Trans. Priv. Secur. (TOPS)* **2018**, *21*, 1–32. [[CrossRef](#)]
33. Ribeiro, J.; Saghezchi, F.; Mantas, G.; Rodriguez, J.; Shepherd, S.; Abd-Alhameed, R. An autonomous host-based intrusion detection system for android mobile devices. *Mob. Netw. Appl.* **2020**, *25*, 164–172. [[CrossRef](#)]
34. Liu, M.; Xue, Z.; Xu, X.; Zhong, C.; Chen, J. Host-based intrusion detection system with system calls: Review and future trends. *ACM Comput. Surv. (CSUR)* **2018**, *51*, 1–36. [[CrossRef](#)]
35. Khadidos, A.; Shitharth, S.; Khadidos, A.; Sangeetha, K.; Alyoubi, K. Healthcare data security using IoT sensors based on random hashing mechanism. *J. Sens.* **2022**, *2022*, 8457116. [[CrossRef](#)]
36. Singh, S.; Sulthana, R.; Shewale, T.; Chamola, V.; Benslimane, A.; Sikdar, B. Machine-learning-assisted security and privacy provisioning for edge computing: A survey. *IEEE Internet Things J.* **2021**, *9*, 236–260. [[CrossRef](#)]
37. Zhang, J.; Chen, B.; Zhao, Y.; Cheng, X.; Hu, F. Data security and privacy-preserving in edge computing paradigm: Survey and open issues. *IEEE Access* **2018**, *6*, 18209–18237. [[CrossRef](#)]
38. Greco, L.; Percannella, G.; Ritrovato, P.; Tortorella, F.; Vento, M. Trends in IoT based solutions for health care: Moving AI to the edge. *Pattern Recognit. Lett.* **2020**, *135*, 346–353. [[CrossRef](#)]

**Disclaimer/Publisher’s Note:** The statements, opinions and data contained in all publications are solely those of the individual author(s) and contributor(s) and not of MDPI and/or the editor(s). MDPI and/or the editor(s) disclaim responsibility for any injury to people or property resulting from any ideas, methods, instructions or products referred to in the content.