STALK: Security Analysis of Smartwatches for Kids

Christoph Saatjohann
Münster University of Applied Sciences
Germany

Fabian Ising
Münster University of Applied Sciences
Germany

Luise Krings
Münster University of Applied Sciences
Germany

Sebastian Schinzel
Münster University of Applied Sciences
Germany

Abstract

Smart wearable devices become more and more prevalent in the age of the Internet of Things. While people wear them as fitness trackers or full-fledged smartphones, they also come in unique versions as smartwatches for children. These watches allow parents to track the location of their children in real-time and offer a communication channel between parent and child.

In this paper, we analyzed six smartwatches for children and the corresponding backend platforms and applications for security and privacy concerns. We structure our analysis in distinct attacker scenarios and collect and describe related literature outside academic publications. Using a cellular network Man-in-the-Middle setup, reverse engineering, and dynamic analysis, we found several severe security issues, allowing for sensitive data disclosure, complete watch takeover, and illegal remote monitoring functionality.

CCS Concepts

• **Security and privacy → Mobile and wireless security**; **Web protocol security**; **Web application security**; Software reverse engineering; • **Social and professional topics** → Privacy policies; • **Human-centered computing → Mobile devices**.

Keywords

smartwatch, web security, certificate pinning, GDPR, wearables, privacy, location tracking, remote listening, GSM

ACM Reference Format:
Christoph Saatjohann, Fabian Ising, Luise Krings, and Sebastian Schinzel. 2020. STALK: Security Analysis of Smartwatches for Kids. In *The 15th International Conference on Availability, Reliability and Security (ARES 2020), August 25–28, 2020, Virtual Event, Ireland*. ACM, New York, NY, USA, 10 pages. https://doi.org/10.1145/3407023.3407037

1 Introduction

Modern embedded computers offer substantial computing power and a variety of wireless interfaces for an affordable price. Because of this, there is a broad range of *wearable* devices, including smartwatches for children with tracking capabilities. These smartwatches offer – among others – location tracking, phone calls, and to take pictures. Parents can use a smartphone application connected to a smartwatch to track their kids or to communicate with them. The kids' smartwatches usually do not directly interact with the app, but the central server provided by the smartwatch vendor relays messages in a store-and-forward manner.

From a security and privacy perspective, the data collected by and exchanged between the smartwatch and app is highly sensible. Compromising this data would mean that an attacker can locate affected kids at any time or that they can read, modify, or delete messages sent from the kids to their parents and vice versa.

One would assume that vendors of kids smartwatches take security and privacy very seriously and make sure that no security vulnerabilities slip into a product that kids use. While there currently are no peer-reviewed publications, several blog posts and reports from researchers describe specific security vulnerabilities in several kids smartwatch products [2, 3, 10, 13].

In this paper, we give an overview of kids' smartwatches available on the market. We then select those watches using a central backend to store and forward messages among kids' smartwatches and parents' apps and perform a structured security analysis of them. The watches are the StarlianTracker GM11, the Polywell S12, the JBC Kleiner Abenteurer, the Pingonaut Panda2, the ANIO4 Touch, and the XPLORA GO. The focus is on the communication between smartwatches and vendor backend (watch-to-backend) that is usually done via GSM and the interaction between parents' apps and vendor backend (app-to-backend), which uses the smartphone's Internet connectivity. Furthermore, we analyze the security of APIs that vendors offer for smartwatch and app communication. To our knowledge, this paper describes the first structured security analysis of the most widely used smartwatches for children available.

The results show that modern kids smartwatches contain critical security vulnerabilities that attackers with very little knowledge of their victim can exploit. We found that an attacker can spoof the position of a watch on three out of the four tested platforms and can spoof voice messages from the watch on two of them. Additionally, an attacker can perform a complete takeover on at least one of the platforms, allowing them to track victims. We also found several privacy problems with the watch platforms.

1.1 Related work

While no peer-reviewed publications on the security of children's smartwatches exist, several authors published substantial work in penetration test reports, blog posts, and talks. In 2017, Forbrukerrådet – the Norwegian Consumer Council – in cooperation with

ARES 2020, August 25–28, 2020, Virtual Event, Ireland

ACM ISBN 978-1-4503-8833-7/20/08…$15.00
https://doi.org/10.1145/3407023.3407037

Mnemonic published the most thorough report available [10]. This report took a look at four smartwatch models for children: Gator 2, Tinitell TT1, Viksfjord – a 3G Electronics watch – and Xplora – an older model of the watch tested in this paper. Forbrukerrådet analyzed these watches for privacy concerns as well as functional security. However, the functional security section of the report is heavily redacted, and despite statements in the report, technical details remain unpublished. Just from the unredacted descriptions in the report, the researchers found several vulnerabilities similar to those we identified, including location spoofing, covert account and watch takeover, misuse of the voice call functionality, and sensitive data disclosure. Upon our request, the researchers denied publishing the details of their research.

Most other reports on smartwatches for children focus on one of the attack surfaces presented in this paper. In 2019, Tod Beardsley et al. analyzed three smartwatches – all of which turned out to be 3G Electronics products – for vulnerabilities in the SMS communication interface [2]. They found that they could bypass the SMS filter and use the undocumented default password to configure and takeover 3G electronics smartwatches[1].

In 2018 and 2019, Christopher Bleckmann-Dreher found several security vulnerabilities in GPS watches manufactured by Vidimensio, which use the AIBEILE backend [3, 18]. These vulnerabilities include tracking of arbitrary watches, wiretapping, and location spoofing. In 2019, researchers from Avast independently discovered the same weaknesses in 29 models of GPS trackers manufactured by Shenzen i365 Tech – a white-label manufacturer using the AIBEILE backend [13]. Similar vulnerabilities were found in November 2019 by Morgenstern et al. for a smartwatch produced by Shenzhen Smart Care Technology Ltd. [15].

In 2019, the European Commission even issued a recall for smartwatches produced by the German manufacturer ENOX because of unencrypted communication and unauthenticated access to data as well as wiretapping functions [5].

1.2 Responsible Disclosure

We disclosed all vulnerabilities in this paper to the vendors with a standard 90-day disclosure deadline and supported them in developing fixes. JBC, ANIO, Pingonaut, and 3G Electronics were very cooperative and provided feedback on our disclosure.

2 Background

2.1 Mobile Cell Phone Communication

All currently available cell phone technologies use encryption algorithms to secure over-the-air communication. Whereby UMTS and newer standards require mutual authentication, the authentication for GSM and GPRS is done on the base station side only. Consequently, the mobile device cannot differentiate between a base station of a valid GSM provider and a rogue station, operated by an attacker. In the latter scenario, the attacker can disable the encryption by providing only the no-encryption mechanism during the pairing.

[1]We found that this default password is documented on several websites as well as in some vendors manuals.

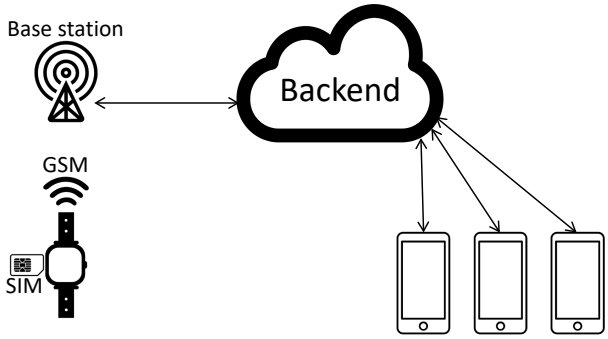

Figure 1: Overview of the communication model of children's smartwatches.

To successfully mount an attack via a rogue base station, the attacker must force the mobile device into the GSM mode and convince it to connect to their station. This is possible by jamming frequencies used by the regular UMTS and LTE networks and, at the same time, provide a GSM network with the best signal strength for the device-under-attack. In such a case, conventional mobile devices will automatically downgrade the connection to GSM and connect to the network with the highest signal-to-noise ratio.

The encryption schemes used for mobile cell phone communication are also well analyzed, and several attacks were published. For the original 2G encryption scheme called A5/1, there is a long history of publicly known research leading to near-real-time decryption of a passive sniffed stream via rainbow tables [4]. The newer A5/3 algorithm, now used for current 2G and 3G communication, also has some vulnerabilities and is considered to be broken [8]. Even the most current cell phone network, LTE, uses encryption mechanism with vulnerabilities that violate typical security objectives like confidentiality or authenticity [16, 17].

2.2 Smart Wearable Devices

Due to the development of increasingly small, low-power embedded microprocessors, more and more formerly unconnected devices are equipped with Internet-of-Things (IoT) technology, forming new smart wearable product categories. These are, for example, life-saving implantable pacemakers, where the newest models communicate via Bluetooth with the patients' smartphone and allow them to track vital data like the daily activity level, directly measured by the pacemaker [14]. The trend is to be more connected and to collect more information about our self and try to optimize our lifestyle.

This trend does not stop with self-optimization but is also present in the tracking of pets, cars, other people – often illegally in many countries – and children. Smartwatches are one way of tracking a child's location and establishing a communication channel between children and parents that has become popular.

Smartwatches for Children Due to the need for a remote and semi-permanent connection, Smartwatches for children are not directly paired to a mobile phone via Bluetooth. Instead, they have their own SIM card and connect to a backend server via GSM. This communication model can be seen in Figure 1.

The functionality of a smartwatch is similar to that of a mobile phone with certain limitations. The main functions are taking photos, sending and receiving voice messages, and making phone calls

– usually only to and from specified contacts, calls from unknown numbers are typically blocked. Another standard function is sending SOS messages to the parents by pressing an SOS button on the watch that triggers an alarm signal with the current location on the parents' smartphone. With the corresponding smartphone application, parents can display the location history, see the current location, take and download pictures, and write and receive messages. Any configuration changes to the smartwatch, e.g., changing the stored contacts or set up geofences, are made with this app.

2.3 Security Marketing of Kids Smartwatches

Smartwatches for children are often offensively marketed with explicit promises of high privacy and security standards. Some manufacturers make explicit promises regarding encryption between watch and server and application and server [19, 20]. One German manufacturer even claims that many watches sold by competitors – some brands are explicitly named – use Chinese infrastructure where personal data is stored in China or Korea [11].

Additionally, multiple vendors explicitly state that their watches do not have a remote monitor functionality – because it would be illegal under various jurisdictions. This marketing promise is, however, severely contradicted by third-party apps available in app stores – i.e., the FindMyKids application [9] – advertising this functionality on the same platform.

Since all of these watches come with a promise of a security gain for both parents and children, it is paramount to take a look both at already published as well as new security vulnerabilities.

3 Attacker Model

The networking model of systems analyzed in this paper consists of the watch, a backend system, and a mobile phone with an app. The watch contains a SIM card and uses the GSM network to connect to the backend system. The app uses the internet connection of the mobile phone, such as GSM or WiFi, to connect to the backend system. The backend system relays messages between watch and app in a store-and-forward manner.

Man-in-the-Middle (MitM) Attacker An attacker that can eavesdrop and modify the connection between either the phone and the backend or the watch and the backend can potentially compromise the security of the communication between these endpoints. Specifically, they are capable of reading and modifying all network traffic between endpoints. This attacker is realistic for the phone to backend communication because users might use untrusted or unencrypted WiFi networks. Additionally, any traffic might be sent over multiple untrusted hosts towards the ultimate destination. This is also true for GSM connections as they are only encrypted between the sender and the base station – which can be impersonated by an attacker. Whether the data is sent encrypted in the backend relies on the GSM provider. A common countermeasure against this attacker is the use of transport encryption, e.g., TLS.

External Attacker - Internet This attacker can connect to any of the servers and endpoints available on the Internet. They are capable of: (1) identifying endpoints used by the watches and the applications (e.g., by reverse engineering or sniffing traffic of their own devices) and (2) sending requests to these endpoints. Generally, these are rather weak attacker capabilities, as anybody can connect to publicly available servers and analyze the traffic of their own devices. Depending on the type of information the attacker wants to access or modify, they might need additional information, for example, phone numbers, device IDs, or usernames, increasing the effort of attacks. This attacker targets a wide range of API weaknesses to circumvent authorization. We restrict our tests in this regard to the most prominent vulnerabilities, including authorization bypass, injection attacks, and insecure direct object reference.

External Attacker - SMS This external attacker can send SMS text messages to smartwatches with GSM capabilities with the intent to execute commands. Specifically, they need to be capable of: (1) finding out the smartwatches' phone number, (2) sending text messages. We assume that an attacker is always capable of (2), whereas (1) is more difficult to achieve. As children's smartwatches usually do not communicate with other devices but the connected mobile phones, the attacker must be able to either probe possible phone numbers or to learn the phone number differently. This might include one of the other attackers, e.g., through an API call leaking registered phone numbers.

Internal Attacker - Privacy and Compliance Since the communication between parents and kids as well as location data of children is sensitive, we consider mishandling and abusing this data an attack. On the one hand, this means that vendors – as per the General Data Protection Regulation (GDPR) – must clearly state what information they collect, where and for how long it is stored, and where the data is transferred. On the other hand, this also means that owners of children smartwatches must not be able to circumvent privacy laws using the device – especially when explicit rulings exist to forbid this. One example of such a feature is the eavesdropping capability of some children smartwatches, which is illegal in several countries. For example, under German law, benign-looking devices must not be eavesdropping devices [7], also under Illinois state law, the recording of a conversation without the consent of all parties is illegal [1]. We assume that an internal attacker – e.g., overprotective parents – might try to circumvent possible restrictions to these functionalities by using one of the other attacker models. They also might be able to apply open source knowledge from the Internet. For example, we found a list of SMS commands (see Section 5.1.3) for one of the watches online, and sending those commands is possible even for laypeople.

4 Analysis

4.1 Selection of Test Samples

The market offers a wide range of smartwatches for kids in different price ranges. Some of them are so-called white-label products that companies can customize and sell under their brand name. The internal hard- and software is usually not modified. One of the largest white-label manufacturers for kids smartwatches is the company 3G Electronics.

For the customer, the Original Equipment Manufacturer (OEM) of the smartwatch is usually not transparent. One indication is the recommended smartphone application. If this app is distributed by a different company than the watch or used for watches of multiple brands, one can assume that the watch is produced by a white-label manufacturer. However, as we will show in Section 5.3,

an application with the same brand name as the watch does not necessarily indicate in-house development of the smartwatch.

To compare different white-label products, we bought three 3G Electronics watches from different brands. The decision for the three additional watches was based on the marketing promises of the producers. They have in common that they promise extraordinary security and privacy level of their products [11, 19, 20]. At the time of selection, we were not aware of the original manufacturers of these three watches. An overview of our analyzed watches is shown in Table 1.

4.2 Intercepting Smartwatch Traffic

Analyzing the smartwatch to server communication requires intercepting the mobile data connection. We create a custom GSM base station with the Software Defined Radio (SDR) N210 from Ettus Research, and the open-source GSM stack implementation OpenBTS inside a controlled and shielded environment. We use the GPS simulator LabSat 3 from Racelogic to spoof different locations for the smartwatch.

While testing all functionalities of the selected smartwatches, we recorded the traffic between the smartwatch and the server using Wireshark on the OpenBTS host computer.

Reverse Engineering of the Communication Protocol To understand the communication between the smartwatch and the server, we had to reverse engineer the used protocol from the recorded traffic. Three of six analyzed watches use an ASCII protocol with no encryption. See Listing 1 for sample messages of the StarlianTracker GM11 watch. The message starts with the ASCII string *3G*, which corresponds to the white-label manufacturer. *74041XXXX* is the IMEI of the watch, and *0008* is the length of the message. The first sample message sends a heart emoji from the app to the watch. The second message triggers the smartwatch to ring, whereas the third message sends the text *Hallo* to the watch. The last message sends a location update based on GPS coordinates, including the battery status and the currently connected cell phone network. By analyzing the protocol, it is also possible to deduce the original manufacturer of a smartwatch.

Listing 1: Redacted protocol messages between the StarlianTracker GM11 smartwatch and the server.

```
[3G*74041XXXXX*0008*FLOWER,1]
[3G*74041XXXXX*0004*FIND]
[3G*74041XXXXX*0020*MESSAGE,00480061006c006c006f0020]
[3G*74041XXXXX*006E*UD,060220,125030,V,40.143057,N,
     7.3223417,E,0.00,0.0,0.0,0,100,91,811,
     0,00000000,1,0,467,193,530,10,147,0,75.1]
```

Finally, we conducted a detailed security analysis of the smartwatch to server communication and the server API by sending manipulated commands while observing the behavior of the watch and app.

Evaluation of SMS commands Some smartwatches support SMS commands to be able to configure specific settings without the need for an internet connection of the watch. In the assessment of such commands, we inserted a valid SIM card into the smartwatch and sent SMS commands with a standard mobile phone to the watch. We also configured the smartwatches with a mobile number that is used in a mobile phone to analyze incoming SMS for the smartwatch.

4.3 Reverse Engineering Smartphone Apps

Analyzing the app to server communication requires us to take a closer look at the implementation of smartphone applications. Mainly, we looked at the Android apps, as this platform allows for more powerful reverse engineering. Where applicable, we also checked the corresponding iOS for the same vulnerabilities. To analyze Android applications, we used the *Frida* reverse engineering framework for hooking function calls and *mitmproxy* for intercepting TLS traffic from both Android and iOS applications. We also used *JADX* to decompile the `.apk` files. The dynamic analysis was performed using a Google Nexus 5X.

Listing 2: Setting up a proxy with Frida.

```
var builder = Java.use('okhttp3.OkHttpClient$Builder');
var proxy = Java.use('java.net.Proxy');
var proxyType = Java.use('java.net.Proxy$Type');
var iSockAddr = Java.use('java.net.InetSocketAddress');
var sockAddr = Java.use('java.net.SocketAddress');
var sa = Java.cast(iSockAddr.$new(IP, PORT), sockAddr);
var type = proxyType.valueOf("HTTP");
var pr = proxy.$new(type, sa);
builder.proxy.overload("java.net.Proxy").implementation =
    function(a) {return this.proxy(pr);}
```

Sniffing TLS encrypted Traffic The first goal was to be able to sniff the traffic between the apps and the backend by installing a proxy server. Since some apps – i.e., SeTracker – explicitly circumvent the usage of the Android system proxy using the OkHttp3 API, we used Frida to hook the constructor of the inner class `okhttp3.OkHttpClient.Builder`, setting up a proxy for requests, as shown in Listing 2. Since this Builder is used to construct all OkHttp client objects, all HTTP and HTTPS requests made through this API will be sent to *mitmproxy* running at `IP:PORT`. Another challenge here lies within the usage of certificate pinning for TLS connection. If an app uses certificate pinning, *mitmproxy* cannot decrypt the redirected traffic. Fortunately, Frida scripts to disable certificate pinning for specific apps exist – i.e., by Jamie Holding [12]. After this, we were able to decrypt all relevant traffic using *mitmproxy*.

Listing 3: Hooking an MD5 method call using Frida.

```
sU = Java.use('com.tgelec.securitysdk.config.SignUtils');
sU.MD5.overload("java.lang.String").implementation =
function(a) {
    var b = this.MD5(a);
    console.log("[+] MD5 of " + a + " is " + b);
    return b;
}
```

Reverse Engineering API Calls After sniffing and decrypting the app to server communication, the next goal was to send crafted API calls to perform in-depth security tests. However, for some API calls encountered during the analysis, information exceeding the simple request was necessary. For example, the *SeTracker* app appends a parameter called `sign` to each request. Since the app dynamically generates this parameter, decompiling the *.apk* file was necessary and hooking the MD5 method call, as shown in Listing 3.

Using *JADX*, we recovered some of the source code of the tested apps. With this information, we could identify additional API endpoints that were not used by the apps during our tests as well as information regarding the API usage. While the decompiler could

Brand	Model	Original Equipment Manufacturer	Android Application
StarlianTracker	GM11	3G Electronics Co., Ltd.	SeTracker, SeTracker2
Polywell	S12	3G Electronics Co., Ltd.	SeTracker, SeTracker2
JBC	Kleiner Abenteurer	3G Electronics Co., Ltd.	SeTracker, SeTracker2
Pingonaut	Panda2	Guangdong Appscomm Co., Ltd.	Pingonaut
ANIO	ANIO4 Touch	3G Electronics Co., Ltd.	ANIO
XPLORA	XPLORA GO	Qihoo 360 Technology Co., Ltd.	XPLORA 3 & 3S

Table 1: Tested watches and corresponding applications as recommended by the product manual.

not recover all source code, we were at least able to identify classes and method signatures. Using Frida, we were able to hook interesting methods and identify inputs and output. This information assisted in the analysis of the API calls.

5 Evaluation

This section summarizes our findings for the tested watch platforms. Since three watches operate with the SeTracker app and the 3 Electronics platform, the corresponding results are combined in one section. Even though the ANIO watch is also manufactured by 3G Electronics, the smartphone app is different, and we will show that the underlying platform is slightly extended (see Section 5.3). The evaluation results of the watch to backend communication can be seen in Table 2, the results of the app and API evaluation can be seen in Table 3.

5.1 SeTracker / 3G Electronics

Listing 4: Text-based protocol used by the StarlianTracker GM11 smartwatches, manufactured by 3G Electronics.

```
[3G* <ID> * <length> * <tag>,<param1>,<param2>,..]
```

5.1.1 *Watch to Backend*

Communication Security All three of the analyzed 3G Electronics platform smartwatches communicate via TCP/IP and a non-standardized protocol. The analysis shows that the protocol used by the first watch, the StarlianTracker GM11, is based on ASCII commands whereby each protocol message is encapsulated in brackets and includes the protocol identifier *3G*, the device ID, the length in bytes of the command, and the command itself. The command consists of a tag with optional parameters separated by commas. An asterisk separates the different elements. The format is shown in Listing 4. This protocol does not use any encryption or authentication mechanism. The security relies only on the underlying GSM network layer.

The next two watches listed in Table 1 communicate via a different –binary– protocol with the server. We did not analyze this protocol since the server also accepts text-based protocol messages for these two watches.

API Security Instead of authentication, the API requires identification by the device ID. The ID of a smartwatch is derived from the International Mobile Equipment Identity (IMEI) number and consists of 10 digits. It is also encoded inside the so-called registration number required for the initial pairing of the watch and the server. Consequently, an attacker who wants to attack a specific smartwatch has to find out the IMEI or the registration ID, which is usually printed on the backside of the watch. During our research, we also found several IMEIs and registration IDs on Amazon ratings and YouTube testimonials.

Our tests show that it is possible to check if a device ID is assigned to a smartwatch and currently paired with a phone. In case the attacker sends a message containing an unassigned watch ID to the server, the server responds with a corresponding error message. If the provided ID was already registered, but later unpaired from the app, the server responds with the email address and, if stored, the avatar image of the last app user who paired the smartwatch with their smartphone. By iterating the device ID, an attacker can scan the server for active IDs, email addresses, and user icons.

Due to the lack of an authentication process, an attacker can send arbitrary messages to the server and impersonate one or more smartwatches by reference to the device ID. It is possible to tamper the data, which the app displays. This includes but is not limited to:

- Modifying the shown location of the smartwatch
- Sending voice messages to the app
- Changing the displayed battery status, time and date of the last update from the watch

If an attacker forces the StarlianTracker smartwatch to connect to their rogue GSM network, establishing a MitM position, they can additionally send new messages or tamper valid server messages to the smartwatch. This adds, at least, the following attack vectors:

- Send voice and text messages to the smartwatch
- Modify SOS and phone book contacts of the watch
- Initiate a hidden call from the watch (Remote Monitoring)

5.1.2 *App to Backend*

Communication Security The SeTracker and SeTracker2 Android applications use a REST API over HTTPS to communicate with the application server. These applications were the only apps we tested that employed certificate pinning. The iOS applications also use certificate pinning, which prevented further analysis on that platform. However, under Android, API calls can still be observed using FRIDA (see Section 4.3). We found that the app adds a `sign` parameter, which is checked server-side, to each API request. The application generates the `sign` parameter by first sorting them alphabetically and then applying the calculation shown in Equation 1. Obviously, this is not a cryptographically strong signature, but merely a measure to obfuscate the request. It is, therefore, not a sufficient security measure to prevent any motivated attacks.

Equation 1: Request signing example. Parameters are loginname=login and password=pass.

$$in = SECPROloginname=<login>password=<pass>SECPRO$$
$$sign = MD5(MD5(MD5(in)))$$

API Security Authentication to the API is achieved by submitting the username and a single-round MD5 hash of the user password to the server. Generally, the use of unsalted MD5 hashes for password hashing is insecure as the hash can be cracked efficiently using

	3G Electronics	Pingonaut Panda2	ANIO4 Touch	XPLORA Go
Encryption	✖	✖	✖	✔
Authentication	✖	✖	✖	–
Active IDs Disclosure	●	●	●	–
Phone Position Tampering	●	◐	●	○
Send Voice Messages to App	●	○	●	◐
Other Vulnerabilities	Sends data outside EU	Disclosure of private data	Location disclosure, Sends data outside EU	

✔ Effective ✖ Not Used/Not working
● Vulnerable ◐ Partly Vulnerable ○ Not Vulnerable – Not Testable

Table 2: Results of the watch to backend communication evaluation

brute force or rainbow tables. Additionally, client-side hashing of passwords does not provide any more security than the use of plain passwords as the hash effectively becomes the password. This is especially true if the password hash is stored on the server and compared to the hash sent by the client to check authorization [6]. The API response contains the MD5 hash of the password, indicating that 3G Electronics stores the password hash on the server. In return to a login request, the API returns a session ID (`sid`), which is used for authentication and which seems to be checked for all relevant calls, in this case preventing unauthorized access to other users' data.

During our tests of the 3G API, we found that almost all parameters of the REST API were vulnerable to SQL injections. Some API endpoints even return the SQL error message and filter parameters containing SQL keywords. Interestingly, while analyzing the Android apps, we found that they employ client-side filtering for SQL keywords. These keywords include typical SQL control sequences like `*`, `select`, `--`, `union`, and `;`. They also include conditional operators like `and`, `or`, and `like`. This filter list, however, is not exhaustive and does not reliably prevent SQL injection exploits. For ethical and legal reasons, we did not exploit this vulnerability to access any data. However, it is safe to assume that a motivated attacker can use this vulnerability to access other user accounts and track arbitrary watches. That 3G Electronics uses keyword filtering shows that they are aware of injection attacks but fail to employ adequate countermeasures – e.g., prepared statements.

Further analysis of the SeTracker and SeTracker2 API reveals additional API endpoints that we did not observe during the traffic analysis. One interesting example is the `sendOrder` endpoint, which is used to send commands to the watch. In particular, two commands stood out. One triggered a 15-second recording on the smartwatch, which can later be downloaded using another endpoint. The watch gives no visual or audible indication that this recording is in progress. The other command allows specifying a phone number to call from the watch. This command causes the watch screen to turn off, and the specified number is called – a remote monitoring functionality. The called number is not restricted to the watch's phone book. In previous versions of SeTracker, this functionality was actively marketed as a *monitor function*. However, we could not trigger this from the app in the current version.

5.1.3 Privacy and Compliance Violations

Smartwatch Communication As far as we could analyze the communication, the communication server for the smartwatch is an Amazon AWS instance located in Frankfurt, Germany. That corresponds to the marketing claims made by the German re-sellers of these watches (see Section 2.3). Since smartwatches of different labels use the same server, we assume that this server is owned and maintained by 3G Electronic, located in Shenzhen, China.

Furthermore, during our research, we found a URL to the management console of the server. Even without valid login credentials, it is possible to see all the functionalities of the console. These functionalities include, but are not limited to:

- Location tracking of smartwatches by the user ID
- List all paired watches for a specific app user
- Reset the password for any app user

Consequently, we can not verify that the data is stored only in Europe, respectively, as remote access is possible.

During startup, the smartwatch opens a connection to a second server, and transmit the following information to it:

- 3G Electronics internal smartwatch platform name
- Device ID and firmware version
- IMEI
- Communication server IP and port
- Mobile network identifier: country and provider
- APN configuration
- Cell phone number of the watch

According to WHOIS information, the server belongs to *Aliyun Computing Co. Ltd.*, a subsidiary of the Chinese *Alibaba* group.

First of all, at least the cell phone number is personal information that is affected by the GDPR. Furthermore, with the transmitted data, it is possible to conduct all the attacks described above.

App Communication The SeTracker and SeTracker2 apps do not provide a privacy agreement upon installation or login. Therefore, users cannot ascertain what data the app will gather and how it will be stored and used. Since the watches used with this application are white-label products, it might be the responsibility of the actual watch vendors to provide privacy statements – which was not the case for our watches. Nevertheless, it seems questionable that an application collecting and processing sensitive data on children does not provide a detailed privacy statement.

During our analysis of the Android application, we found that the exact location (latitude and longitude) of the Android phone is periodically disclosed to the API server to retrieve a value called `adInfo`. We assume this is to deliver advertisement – of which there is plenty in the app – to the client. Since this is not indicated to the user, who is only presented with an Android location permission request upon first starting the app, we find this worrying.

	SeTracker		Pingonaut	ANIO watch	XPLORA 3 & 3S
	SeTracker	SeTracker2			
App Version (Android)	4.5.4	2.6.5	1.10.1	1.1.8	1.8.6.25761
App Version (iOS)	Could not be tested		1.9.0	2.5.1	1.7.7
Encryption	✔		✔	(✓)	✔
Certificate Pinning	✔		✖	✖	✖
Authorization	✔		✔	✔	✔
Authorization Bypass	○		○	●	○
Remote Monitoring	●		○	○	○
Phone Position Disclosure	●	–	–	–	–
Other Vulnerabilities	SQL Injections, MD5 hashed Passwords, Client-side password hashing		–	Forced Browsing No certificate check	RC4 encryption vulnerable to known-plaintext attacks

✔ Effective (✓) Used (with issues) ✖ Not Used/Not working
● Vulnerable ◐ Partly Vulnerable ○ Not Vulnerable – Not Applicable

Table 3: Results of the application and API evaluation

SMS Communication We found several SMS commands for 3G Electronics watches listed on websites and forums[2]. The watch requires a password sent along with the command to execute it. We could partly confirm the vulnerabilities found in [2], all our watches are delivered with the default password *123456*, and only one manual recommends to change this password.

During our evaluation, we found out that the monitor function, which should start a hidden call to a specific phone number from the watch, was not implemented as an SMS command in any of our watches. For the StarlianTracker and the JBC watches, it was possible to activate the automatic answer function. After this, it is possible to call the watch with a muted phone, and the only indication that the watch records the environment is the display, which shows the active call. In our opinion, this functionality is close to a remote monitor function because it is doubtful that the kid, or any other person near the watch, is continuously observing the display. The behavior of the Polywell watch was a bit different. Before the watch answers the call, the ringtone is played for one second, notifying any person around the smartwatch.

With another command, it is possible to pair a smartwatch with a different server. After the execution of this command, the watch will communicate with the newly set IP and port. Due to this mechanism, it is possible to provide an alternative server that will enable the use of third-party applications. At least one of these apps explicitly advertises a remote monitoring function [9] that we successfully tested with the StarlianTracker watch. We were not able to use the app for the Polywell and JBC watches because both communicate via the 3G binary protocol, which is not supported by this third-party app.

5.2 Pingonaut Panda2

Listing 5: Text-based protocol used by the Panda2 smartwatches. N/A indicates an unknown protocol field.

```
#@H<N/A>@H#; <IMEI>; <N/A>; 862182; <command>; <param1>;
    <param2>; ..
```

5.2.1 Watch to Backend

Communication Security The Pingonaut Panda2 smartwatch communicates via TCP/IP and a non-standardized protocol with the server. The text-based protocol contains some parameters which we could not decipher. However, we identified the essential parts of the protocol to tamper messages and to send newly crafted messages. The layout is shown in Listing 5.

The protocol is not encrypted and relies solely on the security of the underlying cellphone network. This is especially interesting because Pingonaut claims that they use a TLS connection between the watch and the server [19].

API Security Our analysis shows that the protocol does not have any authentication mechanism. The identification of the smartwatch is based on the IMEI of the watch. The format of such a 15 digit IMEI is defined as eight digits for the Type Allocation Code (TAC), which is usually unique for a particular product model. The following six digits comprise the serial number of the product, whereby the last digit is a checksum. That means that only six digits – the serial number – are relevant for the Pingonaut Panda2 identification. We confirmed this with a second Panda2, where only these six digits differ.

Pairing a new watch to the app requires a PIN, which is displayed on the watch. We found out that the server sends this PIN to any unpaired IMEI in response to an `init` protocol message. By submitting such requests with invalid IMEIs, more precisely with an invalid checksum, we successfully registered several ghost smartwatches in our app account, which will never be produced. As an attacker, it is possible to pair valid IMEIs of not yet manufactured smartwatches, leading to a Denial-of-Service (DoS) attack on all smartwatches, which will be sold in the future.

Also, this behavior reveals active IMEIs to an attacker. For active IMEIs, the server responds with the following private data if present:

- Unread text messages for the watch
- Speed dial numbers with names stored on the watch
- Do not disturb time intervals
- Stored alarm times

For the analysis of the API security, we replayed messages to the server and crafted new messages. We found that due to the lack of authentication, we could send arbitrary protocol messages to the server. Therefore, an attacker can impersonate any smartwatch. The impersonation attack includes:

- Modifying the shown location of the smartwatch
- Changing the displayed battery status

[2]https://findmykids.org/blog/setting-of-gps-watch-for-kids-using-sms-commands

We could not replay our modified message a few days later, indicating server-side plausibility checks.

During the initial pairing of the smartwatch with the app, the server sends a `Send-SMS` command to the watch. After receiving this command, the watch sends an SMS with the IMEI to the Pingonaut SMS gateway, which in turn acknowledges the phone number of the smartwatch. After this procedure, the user can call the watch from the app. The app will enter the stored number into the dialer app, and the user only needs to start the call.

We found out that the Pingonaut SMS Gateway does not verify the incoming SMS and accepts the sender's phone number as the number of the smartwatch assigned to the IMEI inside the message. In this way, an attacker can silently change the stored watch number for a specific or several IMEIs. In our opinion, the user typically does not verify the called number for the watch and will rely on the stored phone number to call the smartwatch. With this attack, it is possible to redirect phone calls to Pingonaut smartwatches.

Similar to the 3G Electronics watches, in a MitM scenario, the attacker can do the following:

- Send text messages to the smartwatch
- Modify short dial numbers and add numbers which are allowed to call the watch
- Add do not disturb time intervals
- Change alarm times

5.2.2 App to Backend

Communication Security The Pingonaut application uses a REST API over HTTPS to communicate with the backend servers. While TLS is used, no additional authentication of the application server – i.e., certificate pinning – is employed.

API Security User authentication for the Pingonaut API is performed via an API endpoint that takes the username, password, app version, and the app type (*kids*) as parameters. In response, a bearer token – a JSON Web Token (JWT) using the `HS256` algorithm – is returned, which has to be appended to further requests in the Authorization HTTP header. We checked the JWT for common vulnerabilities, including algorithm confusion, leakage of sensitive data, and weak secrets, but where unable to bypass the authorization. A nitpick here is the long (14 days) validity of the JWT, which the server does not invalidate when it issues a new one, causing all tokens to be valid for the whole 14 days. This increases the severity of any possible token disclosure.

All API endpoints (except registration and password reset) require a correct authentication header, and permissions seem to be enforced correctly. Therefore, we were unable to compromise other accounts or devices using the API.

5.3 ANIO4 Touch

5.3.1 Watch to Backend

Communication Security We found that the ANIO4 Touch uses the 3G Electronics ASCII-based protocol, which reveals that it is a white-label smartwatch manufactured by 3G Electronics.

In addition to the known protocol messages, we found one new server response, which the server regularly sends to the watch. We identified the payload of the message as the current weather information, including the name of the nearest city.

API Security Since the watch uses the same protocol, nearly all results listed for the 3G Electronics watches (see Section 5.1.1) are also valid for the ANIO watch. In contrast to the StarlianTracker GM11 watch, an initiation of a hidden call is not possible as a MitM attacker due to firmware modifications made for the ANIO watch. The second limitation is the user icon, which is not supported by the ANIO backend and consequently can not be downloaded by an attacker.

However, due to the weather extension of the protocol, an attacker can send any request to the ANIO server and will get the nearest city of the current smartwatch location.

5.3.2 App to Backend

Communication Security The *ANIO watch* application uses a REST API over HTTPS to communicate with the application server. We found that the app does not check the server certificate and uses no server authentication at all. This is a critical vulnerability, as any active MitM can read and modify the API communication.

Listing 6: Abbreviated response to a request for devices associated with a user ID. Interesting data marked in bold.

```
[{
    "name":"Test Watch","gender":"f","companyId":"3G",
    "phone":"<redacted>","hardwareId":"<redacted>",
    "controlPassword":"<redacted>",
    "lastConnected":"2020-02-25T13:29:17.000Z",
    "id":<redacted>,"anioUserId":<redacted>,
    "lastLocation":{"lng":<redacted>,"lat":<redacted>},
    "lastLocationDate":"2020-02-25T14:29:17.000Z",
    "regCode":"<redacted>",
}]
```

API Security Even though the ANIO watch is a 3G Electronics white-label product, ANIO provides their own API. The `user` endpoint provides registration, login, logout, push token registration, and password change calls. These calls require a valid authorization header – which can be obtained via the login call – where appropriate. The `device` endpoint provides functions for deleting, listing, locating, and configuring devices and is also protected by the same authorization header. Additionally, API endpoints for providing a privacy policy and promotional content exist.

The authorization checks employed by the ANIO API do not prevent a user from accessing other users' data. The server checks if a user is logged in, but does not implement permission management. The only information necessary to access other users' data is the user ID. As IDs are incremented with each new user, they can be iterated by asking the server for user information for any ID. This vulnerability is also known as insecure direct object reference [21].

Using this, an attacker can perform any operation and request any data. This includes locating a watch and viewing the location history, reading and sending chat messages between watch and server, and deleting and registering watches.

Therefore, a complete takeover of watches is possible using this API. A JSON response to a request for devices associated with a user id can be seen in Listing 6. The `lastLocation` parameter reveals the last location the watch reported to the server. The `regCode` parameter, which is also present in the response, is the only information necessary to register the watch with a user ID.

While the ANIO watch is a 3G Electronics white-label product, and the API contains an endpoint similar to the `sendOrder` endpoint

of 3G's API with a `CALL` command, we were unable to trigger a remote monitoring function on the watch.

5.3.3 Privacy and Compliance Violations

Smartwatch Communication According to public WHOIS information, the communication server is hosted on an Amazon AWS instance located in Frankfurt, Germany. The server IP and used ports are not the same as for the 3G Electronics watches. Since ANIO extended the protocol, we assume that they host their own dedicated AWS instance for the ANIO watches.

Due to the weather extension of the protocol, an attacker can request the nearest city of the current smartwatch location. Although this location is not very accurate, this behavior is still a privacy violation.

As described for the other 3G Electronics watches, the smartwatch connects to an update server located in China and sends private data to this server (see Section 5.1.3).

SMS Communication We tested the known SMS commands for 3G Electronics (see Section 5.1.3). Most of them are not implemented in the ANIO firmware. Notably, we could not trigger the remote monitoring and the automatic call answer. Furthermore, setting a different communication server is not possible.

5.4 XPLORA GO

During our analysis of the XPLORA GO watch, we found that the protocol used by both the watch to backend and the app to backend communication was not human-readable. Additionally, the app is highly obfuscated, which makes the analysis challenging.

5.4.1 Watch to Backend

Communication Security The watch communicates with two servers in the backend. The first connection is made via HTTP, the second via a raw TCP connection on port 443. Both connections use Base64 encoded payloads, which seem to be encrypted or at least obfuscated. Based on the analysis of the app in the following, we assume encryption with RC4. We found that only a few bytes change between similar messages. Therefore, we assume the usage of a static key and re-use of the keystream for every message, which would allow known-plaintext attacks.

Due to the URLs called and the size of the messages, we assume that large data items like audio or image files were sent via the HTTP connection.

During our research, the XPLORA GO smartwatch was automatically updated via an over-the-air update. With this firmware update, the HTTP communication was changed to a more secure TLS 1.2 connection to the server. This corresponds to a Blog article from Xplora Technologies in November 2019, which advertises the security and privacy level of the watch, including the usage of TLS between watch and server [20].

API Security In the given time frame, we could not decrypt the protocol messages. Consequently, for the API security evaluation, we tried to replay original messages to the server, which was possible in general, but not in a reliable way. For example, the replay of a voice message with a length of seven seconds from the watch to the app resulted in the display of six voice messages with different lengths – between one and six seconds – inside the app. Despite the different shown lengths, the playback of each of the voice messages reveals the complete audio file. We also found that after truncating the payload of the message, the full voice message is still played. We, therefore, assume that the audio file is stored only once.

Since the device ID is sent in plain as an HTTP request parameter, we replayed messages with tampered IDs. Our analysis shows that the server does not accept the messages for modified IDs. Due to the lack of a second XPLORA smartwatch, we could not determine if this is due to an authentication mechanism or if the tested IDs were not assigned.

5.4.2 App to Backend

Communication Security The app to backend communication is encrypted using TLS with no certificate pinning. By reverse engineering the Android app, we found that any traffic between the app and the backend is additionally encrypted using RC4 inside the TLS stream. The key is derived from specific message values. However, the same key is used for all messages in a session. Also, the keystream is reset for each message. This allows known-plaintext attacks, defeating the RC4 encryption. Additionally, an attacker that can reverse engineer the app can also reverse the key generation algorithm, allowing them to decrypt and spoof messages. However, since the TLS encryption remains unimpaired, we do not see these as critical vulnerabilities.

API Security The application communicates with the backend via a REST API. Every message needs to be RC4 encrypted to be accepted by the server. For encryption of the login request, the app derives a static key in conjunction with a timestamp. Both the user authorization and the selection of the RC4 key are provided by an eight-byte token, which is returned by the login call. Additionally, a ten-digit number (`qid`) – which remains static between sessions – is returned by the login call and is entered into the key derivation:

$$loginkey = MD5(statickey||timestamp)$$
$$session-key = MD5(token||qid)$$

Server and app re-use this key for all messages in a session – until a new login is performed. All API endpoints – except the login – require a valid token, and permissions seem to be checked correctly. Therefore, we were unable to compromise other accounts or devices using the API.

6 Conclusion

During our analysis of smartwatches for children, we found that several of them contain severe security vulnerabilities in either the way the watch communicates with the server, the way the app communicates with the backend, or the way the API on the server is secured. On three out of four tested watch platforms, the impersonation of watches was possible. This allows an attacker to spoof the location of watches. On two platforms, an attacker can even send voice messages from the watch to the smartphone app.

We also found the security of the APIs used by the applications to be concerning. We found critical vulnerabilities (SQL injections and insecure direct object reference) in two of them, allowing complete takeover of watches on at least the ANIO watch platform.

Specifically, the security of the ANIO4 Touch is severely lacking, as the application does not employ certificate checking for the TLS connection to the backend. It was possible to spoof any messages from watch to backend, and the API was vulnerable to

insecure direct object reference, allowing an attacker to track arbitrary watches and eavesdrop on the communication between parents and children.

Examples with better security are the Pingonaut Panda2, where we were unable to identify severe vulnerabilities on the application side, and the XPLORA GO, where we were unable to identify any critical vulnerability.

We were surprised by the small number of OEMs. While we explicitly bought three German premium smartwatches, we found out that these smartwatches were produced by large Chinese OEMs. This is also interesting with regard to financial aspects. While one can buy a 3G Electronics smartwatch for 30 Euro, the same hardware is also sold for 140 Euro advertised as a secure premium product.

All in all, we found the security and privacy of smartwatches for children to be severely lacking and hope that all manufacturers will take the vulnerabilities to heart and fix them to protect children and their parents from attacks. However, as one can see in the ongoing press coverage where vulnerabilities were published for years, public authorities must increase the awareness and take action to force the manufactures to provide secure and legal products.

6.1 Future work

While we had success analyzing the traffic of most tested smartwatches, we experienced problems in analyzing the XPLORA GO watch. Even though we were able to analyze the RC4 encrypted traffic between app and backend through reverse engineering, we were unable to do the same for the watch to backend communication due to time constraints and the absence of watch firmware we could reverse engineer. It might be possible to solve these issues by either obtaining the firmware or mounting a known-plaintext attack against the encrypted traffic.

During our research, we found that some 3G electronics watches – i.e., the JBC watch – use a newer protocol version to communicate with the backend. This protocol is not ASCII-based and, therefore, harder to analyze. However, since the server does not check which protocol version a watch uses, all findings remain valid. Nevertheless, an analysis of this new protocol might lead to new vulnerabilities.

The vulnerabilities we found in children's smartwatches might be present in other devices with similar capabilities such as trackers for cars, pets, or, as sometimes practiced, spouse tracking. Since some companies for kids smartwatches sell such devices as well, we assume that at least some of the vulnerable backends are also used for these devices. This might lead to interesting research directions and the identification of new vulnerabilities and attacker scenarios.

Additionally, looking into other wearable devices, especially devices with medical functionality, might be interesting. Since these devices provide sensitive services and deal with sensitive data, their security is of utmost public concern.

Finally, based on the existing work done in analyzing devices employing GSM communication, other non-wearable devices could be analyzed. This includes medical base stations, IoT devices, and even cars. As all of these devices communicate over GSM with a vendor-provided backend, they should be analyzed for the same vulnerabilities.

Acknowledgments

The authors would like to thank Götz Kappen for providing feedback and hardware for a GPS spoofing setup. Additionally, we would like to thank Hanno Böck for providing valuable feedback during our research.

Christoph Saatjohann and Fabian Ising were supported by the research project "MITSicherheit.NRW" funded by the European Regional Development Fund North Rhine-Westphalia (EFRE.NRW). Fabian Ising was also supported by a graduate scholarship of Münster University of Applied Sciences.

References

[1] Illinois General Assembly. 2014. 720 ILCS 5 - Article 14. Eavesdropping. http://www.ilga.gov/legislation/ilcs/ilcs4.asp?DocName=072000050HArt%2E+14&ActID=1876&ChapterID=0&SeqStart=34200000&SeqEnd=35400000.

[2] Tod Beardsley. 2019. IoT Vuln Disclosure: Children's GPS Smart Watches (R7-2019-57). https://blog.rapid7.com/2019/12/11/iot-vuln-disclosure-childrens-gps-smart-watches-r7-2019-57/

[3] Christopher Bleckmann-Dreher. 2019. Watchgate - How stupid smartwatches threaten the security and safety of our children. Troopers 2019. https://www.troopers.de/troopers19/agenda/yugzay.

[4] Giuseppe Cattaneo, Giancarlo Maio, and Umberto Petrillo. 2013. Security Issues and Attacks on the GSM Standard: a Review. *JOURNAL OF UNIVERSAL COMPUTER SCIENCE* 19 (01 2013), 2437–2452. https://doi.org/10.3217/jucs-019-16-2437

[5] European Commission. 2019. Alert Number: A12/0157/19 - Smart watch for children (ENOX Safe-KID-one). https://ec.europa.eu/consumers/consumers_safety/safety_products/rapex/alerts/?event=viewProduct&reference=A12/0157/19

[6] MITRE Corporation. 2012. CWE-836: Use of Password Hash Instead of Password for Authentication. https://cwe.mitre.org/data/definitions/836.html

[7] Bundesministerium der Justiz und für Verbraucherschutz. 2012. §90 TKG - Missbrauch von Sende- oder sonstigen Telekommunikationsanlagen. https://www.gesetze-im-internet.de/tkg_2004/__90.html.

[8] Orr Dunkelman, Nathan Keller, and Adi Shamir. 2010. A Practical-Time Attack on the A5/3 Cryptosystem Used in Third Generation GSM Telephony. Cryptology ePrint Archive, Report 2010/013. https://eprint.iacr.org/2010/013.

[9] FindMyKids. 2019. FindMyKids GPS Child Tracking App: Features & Benefits. https://findmykids.org/blog/en/gps-child-tracking-app Retrieved: 2020-03-21.

[10] Forbrukerrådet. 2017. #WatchOut - Analysis of smartwatches for children. https://fil.forbrukerradet.no/wp-content/uploads/2017/10/watchout-rapport-october-2017.pdf

[11] ANIO GmbH. 2020. Kinder-Smartwatches: Sicherheitsgewinn oder Sicherheits-Risiko? https://www.aniowatch.com/2020/04/04/kinder-smartwatches-sicherheitsgewinn-oder-sicherheits-risiko Retrieved: 2020-04-13.

[12] Jamie Holding. 2019. Advanced Certificate Bypassing in Android with Frida. https://blog.jamie.holdings/2019/01/19/advanced-certificate-bypassing-in-android-with-frida/

[13] Martin Hron. 2019. The secret life of GPS trackers. https://decoded.avast.io/martinhron/the-secret-life-of-gps-trackers.

[14] Medtronic 2019. *User Guide: My CareLink Heart App.* Medtronic, https://www.medtronic.com/content/dam/medtronic-com/de-de/patients/documents/carelink/mycarelink-heart-app_user-guide_medtronic.pdf.

[15] Maik Morgenstern. 2019. Produktwarnung! Chinesische Kinderuhr verrät tausende Kinder. https://www.iot-tests.org/de/2019/11/produktwarnung-chinesische-kinderuhr-verraet-tausende-kinder.

[16] David Rupprecht, Katharina Kohls, Thorsten Holz, and Christina Pöpper. 2020. IMP4GT: IMPersonation Attacks in 4G NeTworks. In *ISOC Network and Distributed System Security Symposium (NDSS)*. ISOC, San Diego, CA, USA, 15.

[17] D. Rupprecht, K. Kohls, T. Holz, and C. Pöpper. 2019. Breaking LTE on Layer Two. In *2019 IEEE Symposium on Security and Privacy (SP)*. IEEE, San Francisco, CA, USA, 1121–1136. https://doi.org/10.1109/SP.2019.00006

[18] Fabian A. Scherschel. 2018. Achtung, Uhr hört mit. *c't Magazin für Computertechnik* 8 (2018), 062.

[19] Pingonaut Team. 2020. Test bestanden! https://pingonaut.com/de/news/test-bestanden. Retrieved: 2020-03-15.

[20] Xplora Technologies. 2019. Sicher und zuverlässig! https://shop.myxplora.de/blogs/news/sicher-und-zuverlassig Retrieved: 2020-03-15.

[21] The Open Web Application Security Project (OWASP). 2013. OWASP Top 10 - 2013. https://owasp.org/www-pdf-archive/OWASP_Top_10_-_2013.pdf.