electronics

MDPI

Article

How to Circumvent and Beat the Ransomware in Android Operating System—A Case Study of Locker.CB!tr

Kornel Drabent [1], Robert Janowski [1] and Jordi Mongay Batalla [2,*]

[1] Faculty of Informatics, Warsaw School of Computer Science, Lewartowskiego 17, 00-169 Warsaw, Poland; rjanowski@ms.wwsi.edu.pl (R.J.)
[2] Institute of Telecommunications, Warsaw University of Technology, Nowowiejska 15/19, 00-665 Warsaw, Poland
* Correspondence: jordi.mongay.batalla@pw.edu.pl

Abstract: Ransomware is one of the most extended cyberattacks. It consists of encrypting a user's files or locking the smartphone in order to blackmail a victim. The attacking software is ordered on the infected device from the attacker's remote server, known as command and control. In this work, we propose a method to recover from a Locker.CB!tr ransomware attack after it has infected and hit a smartphone. The novelty of our approach lies on exploiting the communication between the ransomware on the infected device and the attacker's command and control server as a point to reverse disruptive actions like screen locking or file encryption. For this purpose, we carried out both a dynamic and a static analysis of decompiled Locker.CB!tr ransomware source code to understand its operation principles and exploited communication patterns from the IP layer to the application layer to fully impersonate the command and control server. This way, we gained full control over the Locker.CB!tr ransomware instance. From that moment, we were able to command the Locker.CB!tr ransomware instance on the infected device to unlock the smartphone or decrypt the files. The contributions of this work are a novel method to recover the mobile phone after ransomware attack based on the analysis of the ransomware communication with the C&C server; and a mechanism for impersonating the ransomware C&C server and thus gaining full control over the ransomware instance.

Keywords: ransomware; Locker.CB!tr; C&C server; reverse engineering; ransomware mitigation; Android; smartphone screen locking; malicious symmetric file encryption

check for updates

Citation: Drabent, K.; Janowski, R.; Mongay Batalla, J. How to Circumvent and Beat the Ransomware in Android Operating System—A Case Study of Locker.CB!tr. *Electronics* **2024**, *13*, 2212. https://doi.org/10.3390/electronics13112212

Academic Editors: Lixin Wang, Qiang Ye and Jianhua Yang

Received: 29 April 2024
Revised: 25 May 2024
Accepted: 4 June 2024
Published: 6 June 2024

Copyright: © 2024 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

During the past 15 years, ransomware has gained much attention among different kinds of evolving malicious software [1]. It extorts victims to pay money to regain access to their data stored in the attached device. Because of the financial motivations of attackers, ransomware often targets the business sector as companies are more willing to pay to regain access to valuable data. However, with the spread of smartphones, the number of ransomware attacks against these devices also increases. The scale of the problem is stimulated by the easily accessible toolkits that allow carrying out a ransomware attack without acquiring complex knowledge about the operating systems, programming languages, and cybersecurity methods [2].

Defense from ransomware attacks is not easy, especially when the damage has been provoked. Some actions can be taken to prevent or detect and mitigate the attack (by diminishing the likelihood or the consequences). However, the high number of attacked systems (many OSs) and the widespread range of attacks make defense very challenging.

In this paper, we propose a new ransomware attack mitigation solution. We focus on Locker.CB!tr ransomware that targets smartphones with Android OS. We provide an in-depth analysis of its structure and operation methods. We disclose the communication

patterns between the ransomware and its command and control (C&C) server, including the sending of commands with malicious actions and receiving their confirmations. Instead of discovering keys in the code, we exploit this knowledge to prepare our own C&C server that would mimic a true attacker's control. Furthermore, using the hooks, we replace the IP address of the attacker's C&C server with the IP address of our own server. As a result, we gain full control over the Locker.CB!tr ransomware instance as the whole communication is redirected to our C&C server in a way undetectable for the ransomware. From that moment, we can command the Locker.CB!tr ransomware instance on the infected device to unlock the smartphone or decrypt the files on it.

The novelty of the research and the authors' contributions are as follows:

- Our approach is to provide the ransomware mitigation solution once it hits the victim, while most of the works in this area deal with detection or prevention against ransomware.
- We provide a procedure for recovering the infected mobile device based on the analysis and impersonation of the communication between the ransomware instance of the terminal and its C&C server.
- We accomplish the reconstruction of the communication patterns and exchanged data formats between the ransomware instance and its C&C server.
- We provide a method to dynamically replace the IP address of the original C&C server with our own. This leads to impersonating the ransomware C&C server and gaining full control over the ransomware instance of the infected mobile device.
- We develop a C&C server implementation that may communicate with the ransomware instance and order it to decrypt the file system and unlock the terminal screen after a ransomware attack.

The rest of this paper is organized as follows. Section 2 analyzes the literature on defense methods against ransomware. Section 3 starts with the overview of Locker.CB!tr ransomware functioning and then presents our analysis and method to circumvent and beat this ransomware. Section 4 verifies and discusses the effectiveness of our solution, and Section 5 summarizes the work.

2. Defense from Ransomware: Analysis of Existing Solutions

In the literature, one can find three groups of defense methods defined with respect to the type of actions undertaken by computer or smartphone owners: detection, prevention, and mitigation.

2.1. Methods for Detecting Ransomware

The aim of the detection type defense is to recognize the threat before it hits the system—usually during the application installation process or early stage of its operation. A typical approach against viruses and malware relies on identifying some characteristic features of the malicious software in a suspected application. These features define the signature of malicious software and are typically collected in a database that comes with anti-virus or anti-malware programs and stores. This approach was proven to be ineffective in the case of ransomware [3]. The main problems are fast-appearing new or re-structured ransomware with non-recognized signatures and using code obfuscation techniques that make the signature useless [4]. For these reasons, new methods have been proposed that can overcome the limitations of the signature-based approach. These are either machine learning (ML) or model-based approaches, with the majority belonging to the first category.

The machine learning approach is very appealing in this context since it can work with many features that describe the application we want to classify as benign or ransomware. Thus, they can be very effective against vast groups of ransomware, even those never seen before (zero-day). The solutions in this category differ in the number or type of extracted features or the machine learning algorithm that was used. For example, the authors of [5] used both static features extracted from AndroidManifest.xml and dynamic ones such as API calls and applied the Random Forest ML algorithm. A work presented in [6] used file access monitoring, detection of encryption (based on Shannon entropy measure), and

real-time layout analysis. The dynamic features extracted from actions like displaying threatening messages, screen locking, or file encryption were used by the developers of HellDroid detection system [7]. The authors of [8] proposed a new ransomware detection software based on dynamic analysis where system calls as features were extracted and used in Random Forest and Naïve Bayes ML algorithms. Scalas et al. extracted classes and methods from dex files that were exploited as the training feature set [9].

On the other hand, model-based approaches are more heterogeneous in the sense of the system characteristics the model is based on. In [10], the authors proposed ransomware detection solely based on formal methods derived from Java bytecode analysis. This formal model was constructed according to the rules of the calculus of communicating systems approach [11] and fully relied on static analysis. Ramesh and Maneen proposed another ransomware detection model based on a dynamic analysis modelled as a Finite State Machine (FSM) [12]. The results of detection effectiveness evaluation were quite high and similar for both approaches. The difference is the required effort for building a model or preparing the training and testing datasets. Nevertheless, some authors argue that detection is not an appropriate method, especially the one that exploits dynamic features either in ML or model-based approaches, since it requires the ransomware to start operating, thus leading to the hazard of too-late decision-making [13]. As a remedy, they propose prevention methods.

As a result of the rising threat from ransomware attacks, security software companies and security development teams started producing antivirus tools capable of detecting and removing malware. Some products are free, such as AVG AntiVirus FREE [14]; some are offered as free or paid (premium) versions, e.g., Avast Mobile Security [15], and some are exclusively commercial, e.g., Bitdefender Mobile Security [16] or Norton 360 [17]. Without experiments, it is difficult to recognize what antiviruses are capable of ransomware detection since their descriptions do not always include such information. The examples of products for which this information is explicit are Mobile Security & Antivirus from Trend Micro company [18], Ransomware Defender from ShieldApps [19], Norton 360 [17], and ZoneAlarm Mobile Security from CheckPoint [20]. The mechanisms for ransomware detection implemented in these antivirus apps are not revealed, nor is their effectiveness. According to VirusTotal [21] (a web portal that is the front end of several tens of antivirus applications), Locker.CB!tr ransomware might be detected by 29 out of 67 available antiviruses. Among the paid software capable of detecting Locker.CB!tr are the products from the following companies: Fortinet, Kaspersky, McAfee, Avast, and Bitdefender. However, Malwarebytes, Avira, and Zillya are counterexamples.

2.2. Methods for Preventing Ransomware

The methods for preventing ransomware rely on the proactive actions implemented in the system by additional software that allow for recovery from the situation provoked by ransomware without the knowledge of its structure or dynamic behavior. The representative examples are PayBreak [22] and ShieldFS [13], although they use different prevention mechanisms. PayBreak hooks the cryptographic functions and collects information about the algorithm and parameters (keys) used for file encryption. When a file is maliciously encrypted, PayBreak's recovery mechanism applies the knowledge gained to decrypt it. This method has a clear drawback since it is based on the prior recognition of the cryptographic libraries used by ransomware. Advanced attacks could use widely popular libraries or create their own ones that do not leave known signatures, making them difficult to detect. The alternative is ShieldFS, which is an add-on driver for Windows OS that implements a copy-on-write mechanism at a low level for all file write operations. ShieldFS may always recover the original file by reverting back to the copy. This defense concept comes at the expense of additional computational and disk space capacity.

2.3. Methods for Mitigating Ransomware

The last group of defense methods focus on ransomware attack mitigation. Since these methods try to recover from the attacks that have already been conducted, they rely on the weaknesses of concrete mechanisms or their implementations. The encryption algorithms attackers use are resistant to cryptoanalysis and thus crackable only by brute-force attacks [23]. As the secret keys used in these cases are long, typically 256 bits or longer [3], it is not feasible that the victims can regain access to their data unless they agrees to fulfil the attacker's requests. Obedient victims regain access to their devices or data once they paid the ransom. In the latter case, an attacker decrypts the data using a secret key known only to them. Therefore, two main mechanisms are implemented in ransomware software: the encryption of the victim's data and a remote control of the compromised device to decrypt the data on command. This is generally carried out from a command and control (C&C) server. Ransomware analysts willing to defeat the malicious software can look for the fragments of code that implement these functions. On the contrary, the attackers, aware of these methods, try to hinder such practices by applying a number of techniques like code obfuscation [24] and incremental unpacking [25]. Also, the communication channel exploited by an attacker to command and control (C&C) the victim's device might be masqueraded to avoid detection. A good example might be a covered channel [26,27] or the usage of social media bots [28] to fool security analysts and protection mechanisms. Despite these impediments, the analysis of the application code or network traffic is the most promising approach to spot ransomware. For this reason, researchers analyze specific ransomware code and try to exploit its vulnerabilities to defeat it. In [29], the authors analyze Avaddon ransomware attack and propose a reverse encryption operation based on the obtention of encryption keys from the memory dumps. The same concept is used in [30], where Bajpai and Enbody report successful decryption after NotPetya, WannaCry, LockCrypt, CryptoRoger, or AutoIT attacks. Kim et al. exploit a weakness of Rhysida implementation (more precisely, the pseudorandom generator used for cryptography key creation) that allows for the regeneration of the decryption key [31]. These conditions suffice to ensure the successful description of the files. The solutions described in [29–31] were prepared for Windows OS; however, the same concept might be applied to other operating systems, including Android OS. The common feature of mitigation solutions is an in-depth analysis of particular ransomware implementation in search of weaknesses. In the context of ransomware mitigation, the work presented in [32] might be considered really valuable as it reveals an in-depth analysis of the consecutive variants of the Jisut ransomware family. It gives good insights into the architecture, implementation, and evolution, which is considered an important contribution in looking for Jisut attack mitigation. The analysis of network traffic content and API calls [5] might also help in beating the ransomware that has already attacked our system and encrypted our data.

In summarizing the research on ransomware defense methods, it is worth emphasizing that there are relatively many works about ransomware detection solutions and relatively small contributions to ransomware attack mitigations. This might come from the fact that the latter require much effort in analyzing the ransomware code and advanced knowledge to exploit its weaknesses.

3. A method to Mitigate the Ransomware—The Case Study of Locker.CB!tr

3.1. Overview of Locker.CB!tr

Most extorting ransomware is commanded by C&C servers through HTTP connections. If this communication is not carried out accurately, it may present several vulnerabilities that may be exploited to order the ransomware instance to decrypt the encrypted files. Locker.CB!tr is a good example of a ransomware instance managed from its C&C server since it maintains an extensive message exchange. Even if Locker.CB!tr might be considered a relatively old ransomware example, let us remark that the main characteristics of the operation that we exploited to recover after an attack (i.e., commanding the malware to reverse malicious actions) are still present in more recent malware. For example, the

BlackRock malware discovered in 2020 [33] is a banking trojan controlled by a C&C server that can command the infected device to perform the Hide_Screen_Lock action to keep the device on the HOME screen and, in this way, prevent its detection by an antivirus. Locking the device on HOME screen can be reversed by the command Unlock_Hide_Screen sent by the C&C server. Another example might be Escobar Android malware, which is also a kind of banking trojan spotted in 2022 [34]. It aims to steal money from a bank account and may even bypass MFA (Multi-Factor Authentication). However, it might be neutralized from the C&C server by the "Kill Bot" command, which simply causes the malware to delete itself. The same method of stopping malicious activities after the infection is possible in the cases of SOVA [35] and Nexus [36] banking trojans discovered in 2021 and 2023, respectively. Both malware apps are controlled by their C&C servers, which may send a "delbot" command that deletes the malware from the device. The reason for implementing auto-destruction commands is the ability to cover one's tracks in case the trojan instance could be detected by an antivirus application or after completing a theft task. In any case, from a defender's or victim's point of view, the communication with the C&C server presents a number of vulnerabilities that are an effective way of recovering after a malware attack.

Locker.CB!tr ransomware targets Android users and spreads by mimicking a useful application—VideoPlayer. It locks the screen and displays a message saying that suspicious, illegal activity has been detected on the phone and threatens that the case will be reported to the FBI [37]. It expects the victim to pay a USD 500 ransom to unlock access to a smartphone. Locker.CB!tr communicates with a C&C server, and depending on what commands it receives from it, it can encrypt files on the SD card, send an SMS, check an incoming SMS, or check the contact list. The malware spreads by sending an SMS to your contact list with a link that redirects you to malware.

Locker.CB!tr fits into the CryptoLocker typology, which is often used by cybercriminals to encrypt files on an infected device and demand ransom from victims. Unlike other forms of ransomware that encrypt files using variables and complex cryptographic algorithms, Locker.CB!tr ransomware uses a static key to encrypt files [38]. In the case we have examined, the key (or rather a string of characters used to derive the key) was hidden in the code as a string of characters in the class responsible for receiving commands from its C&C server. It is not an exceptional case—it was revealed that Jisut ransomware also hard-coded its encryption keys [32]. This ransomware usually gets onto the victim's device through emails with infected attachments or through SMSes with links to malicious websites. Locker.CB!tr is distributed as a legitimate multimedia application under the name of VideoPlayer. Once installed on the victim's device, Locker.CB!tr begins encrypting files such as documents, photos, videos, and other data using a strong AES symmetric encryption algorithm with the secret key stored locally (more precisely, the key is derived from a hard-coded sequence). After the encryption process is completed, a message is displayed on the screen stating that the files have been encrypted, usually to scare the victim into paying a ransom [3]. Once it hits a device, it contacts its C&C server using a hard-coded IP address (56% of ransomware use this method [39]), reports the successful infection, and waits for the commands. It distributes itself by sending SMSes to addresses from the contact list. These SMSes contain the link to malicious VideoPlayer and an incentive to download and install it. Locker.CB!tr maintains an HTTP connection to the C&C server. The fact that a plain text protocol is used for communication purposes instead of its secure variant (HTTPS) is not so surprising. According to the information from [40], more than 99% of Android botnets use HTTP-based web traffic to receive bot commands from their C&C servers. Using this communication, the attacker, after receiving the ransom, can unlock access to the terminal interface and decrypt files, thus allowing the victim full access to the compromised terminal. The list of commands executed by Locker.CB!tr ransomware is provided in [38].

3.2. Methodology and Materials

Our research methodology assumes the investigation of a selected ransomware, Locker.CB!tr, including its full disclosure, understanding its behavior and weak points that can lead to ransomware mitigation after it infects a device. We assume that the findings from the investigation stage of our research will allow us to work out an approach that can be generalized for a wider group of ransomware. We start the investigation by re-creating Locker.CB!tr source code in the process of reverse-engineering. Through static and dynamic analysis, we try to disclose the full communication stack between a ransomware instance on an infected device and its C&C server. As one of the first elements of communication analysis, we try to find out if the IP address of the C&C server is locally stored or is determined in a dynamic way using Domain Generation Algorithms (DGAs) [39]. Then, we reconstruct the communication pattern and data formats exchanged between the ransomware and its C&C server. On the basis of the recognized communication pattern, we prepare our own C&C server and redirect the ransomware communication by replacing the original IP address with our own during the ransomware runtime. In this way, we impersonate the C&C server and gain full control over the ransomware instance on an infected device. Finally, we command the ransomware instance, including unlocking the smartphone screen or decrypting the files.

A key element in malware analysis is an environment that will protect sensitive data while performing experiments with malicious software, especially ransomware. Such an environment is provided by an Android Virtual Device (AVD) emulator built into Android Studio IDE and Android Debug Bridge (ADB)—a client-server program used to communicate with an Android device that provides access to Unix shell [41]. The use of an emulator instead of a real device is a common practice; for example, in [8], the authors use the Genymotion Android emulator. The set of tools used for the analysis can be divided into two groups: static and dynamic. The tools used for static analysis are as follows:

- Enjarify—a tool similar to "dex2jar". It converts Dalvik bytecode to Java bytecode so that further analysis can be based on Java decompilers, e.g., CFR, Fernflower.
- Apktool—it is an open-source software that allows the conversion of application resources such as binary XML into plain text, as well as decompiling the executable code of the Dalvik machine into an intermediate format called smali.
- ByteCodeViewer—it is a multifunctional software that also serves as a decompiler, editor, and even debugger. The main advantage of this tool is the ability to divide the view of the decompiled class into many parts and use different decompilers on each part. It allows us to compare the performance and effectiveness of different decompilers and choose the best one in a given case. Currently supported decompilers are Procyon, CFR, JD-GUI, FernFlower, Krakatau, and JADX. Besides that, it also allows the display of the content of Smali files or Java bytecode.

The tools used for dynamic analysis are as follows:

- Frida—a set of tools for dynamic code manipulation that allows for injecting fragments of JavaScript code or own libraries into native applications on Windows, Mac OS, iOS, and Android. More specifically, this tool provides the ability to analyze the behavior of the program at runtime by injecting code into the target process.

3.3. Analysis of Locker.CB!tr Ransomware Operation

The analysis of Locker.CB!tr ransomware started with the application of Apktool to decode the apk archive to obtain the AndroidManifest.xml file (Figure 1), which revealed the required permissions and registered intent filters [42]. In such analysis, the most attention is paid to the presence of BOOT_COMPLETED intents (visible in section <intent-filter> in Figure 1), where the authors of ransomware usually hide the start code of the malware [43].

```
<receiver android:name="com.adobe.videoprayer.CheckerBroadcastReceiver"/>
<service android:exported="false" android:label="@string/app_name" androic
<service android:exported="false" android:label="@string/app_name" androic
<receiver android:enabled="true" android:exported="true" android:label="@s
    <intent-filter android:priority="2147483647">
        <action android:name="android.intent.action.BOOT_COMPLETED"/>
    </intent-filter>
</receiver>
```

Figure 1. The content of AndroidManifest.xml file from the VideoPlayer application with Locker.CB!tr hidden inside.

The next step was the conversion from Dalvik bytecode (dex) to Java bytecode (jar) accomplished with the help of the enjarify tool. The conversion to Java bytecode made it possible to open the code with ByteCodeViewer and analyze the recreated structure of classes with available decompilers. The most promising results of this process, in the sense of clear Java code, were obtained with CFR and Fernflower decompilers. Despite the fact that parts of the code were obfuscated, as indicated by the warnings from ByteCodeViewer during the decompilation of the AbsRequest class, most of the code was decompiled to a form of high readability. This, in turn, allowed us to freely move around this and other classes and understand most of them, including the commands Locker.CB!tr ransomware supports in communicating with its C&C server.

One of the main classes responsible for communication between the Locker.CB!tr instance on the infected device and its C&C server is the AbsRequest class (Figure 2). The code of the AbsRequest class reveals the IP address of the ransomware C&C server since it is hard-coded there without any obfuscation. However, according to [44], it is not the exception but one of three typical methods of storing the IP address of the C&C server on the ransomware side. Furthermore, the code of the AbsRequest class parses the responses from the C&C server (in parseNetworkResponse method—Figure 3) using the GSON library, which suggests that information from the C&C server is transferred in JSON format.

```
public abstract class AbsRequest extends JsonRequest {
    public static String BASE_URL = "http://148.251.154.104:12449";
    private static final String LOG_TAG = "AbsRequest";
    private static final int TIMEOUT;
```

Figure 2. The result of the decompilation: the readable part of the Java code of the AbsRequest class.

```
protected Response parseNetworkResponse(NetworkResponse networkResponse, Class clazz) {
    Object object;
    int n = networkResponse.statusCode;
    int n2 = 200;
    if (n == n2) {
        try {
            object = new Gson();
            byte[] byArray = networkResponse.data;
            String string = "UTF-8";
            String string2 = new String((byte[])byArray, (String)string);
            object = object.fromJson((String)string2, clazz);
            n2 = 0;
            string2 = null;
            object = Response.success((Object)object, null);
        }
        catch (Exception exception) {
```

Figure 3. The result of the decompilation: parseNetworkResponse method from the AbsRequest class, which is responsible for parsing the response from the C&C server.

The analysis of AbsRequest class code revealed that messages sent by the malware to its C&C server are encoded in a format consistent with application/x-www-form-urlencoded MIME type, i.e., all data are stored in the URL address as an appropriately formatted string in which the parameters and their values are encoded in parameter-value tuples separated by '&', with a '=' between the parameter and its value. The AbsRequest class, which is the base class defining queries sent from the infected terminal to the C&C server, together with other classes inheriting from it, e.g., the DeviceDataRequest class, formats data sent to the C&C server in accordance with the MIME type application/x-www-form-urlencoded. Specific classes are used with appropriately crafted URLs depending on the type of data being transferred. These URLs inform the C&C server about the type of data and information that Locker.CB!tr is willing to relay. The list of class names for specific URLs and their meanings are provided in Table 1.

Table 1. The list of specific URL additions representing specific types of information transferred from the infected terminal to the C&C server.

Specific URL Element	Name of Implementing Class	Meaning
/eaction/	CommandConfirmRequest	Confirmation of the previous command completion
/gac/	CommandRequest	Request for the next command
/sc/	ContactsRequest	List of contacts read from the infected terminal
/pha/	DeviceDataRequest	Information about the infected terminal
/cpm/	PaymentRequest	Information about the payment (ransom)
/scs/	ReceivedSmsRequest	Information about the SMS captured at the infected terminal (content and delivery information)
/ssms/	SmsRequest	List of SMSes read from the infected terminal
/gfsf/	SmsToSendRequest	Request for data to send an SMS from the infected terminal (SMS destination address and content)
/gt	UserAddressRequest	Request for information about the country, city, and IP address of the infected terminal
/logsms/	SuccessCountRequest	Information about the number of SMSes successfully sent from the infected terminal

Commands from the C&C server to the infected terminal are transmitted in HTTP responses to queries sent by Locker.CB!tr ransomware. This method of initiating communication between the ransomware and the C&C server allows one to bypass possible protection by firewall systems. This is due to the fact that HTTP communication is usually open in the direction from a terminal to the Internet, but in the opposite direction (from the Internet to the terminal), only responses to previously sent queries pass.

The reception of the commands from the C&C server is implemented in the AbsRequest base class and derived classes in the parseNetworkResponse method. The declaration of the parseNetworkResponse method in the AbsRequest class contains two parameters of the type "NetworkResponse" and "Class". In derived classes, the implementation of the parseNetworkResponse method contains either the same two parameters or only one—the one of the "NetworkResponse" type. The second parameter specifies the name of the class, defining the storage format for data read from the JSON received in the response from the C&C server. This means that the data transferred in the HTTP response from the C&C server in JSON format will be saved in an object of the class type, which is the second argument of the parseNetworkResponse method call. For example, the definition of the parseNetworkResponse method from the SmsToSendRequest class indicates the SmsToSendData class as the second parameter of the call. The definition of the SmsToSendData class includes the format for storing data related to the order in which SMS messages are sent. To perform such an action one needs the SMS recipient's number and the content of the SMS message. Therefore, the definition of the SmsToSendData class contains the field mData of the SmsToSendInnerData type, which collectively stores information about the SMS recipient's number and its content in the mPhone and mSms fields, respectively (Figure 4).

```
public class SmsToSendData {
    SmsToSendInnerData mData;

    public SmsToSendInnerData getData() {
        return this.mData;
    }
}
```

```
public class SmsToSendInnerData {
    String mPhone;
    String mSms;

    public String getPhone() {
        return this.mPhone;
    }

    public String getSms() {
        return this.mSms;
    }
}
```

Figure 4. The code of the SmsToSendData class (on the left) and the definition of the SmsToSendInnerData type used (on the right).

After receiving and parsing the response from the C&C server, the stage of interpreting and reacting to the received command begins. The CheckerService class is responsible for responding to commands from the C&C server, which is an Android service (implemented as an IntentService class) running in the background. From the analysis of the source code of the CheckerService class, it can be concluded that after starting the operation (in the onHandleIntent method), Locker.CB!tr collects data about the device, including device ID, phone number, and software version, by indirectly calling the getAndSendDeviceData() method from the makeActions() method. The collected data are saved in the form of an object in the DeviceDataRequest class. Additionally, the CheckerService class implements methods corresponding to actions performed by Locker.CB!tr. The set of supported actions is defined in the ActionType class. After recognizing one of the defined action names, Locker.CB!tr executes the appropriate code fragment that implements the related action. One such action is ACTION_LOCK_PHONE ("case 5:" in the conditional structure in Figure 5), which results in locking the user's screen, thus preventing further use of the device.

```
case 1: {
    object3 = "VideoPrayer";
    object2 = "I should send SMS";
    Log.e((String)object3, (String)object2);
    this.getAndSendSMSData();
    break;
}
case 2: {
    object3 = "VideoPrayer";
    object2 = "I should send contacts";
    Log.e((String)object3, (String)object2);
    this.getAndSendContactData();
    break;
}
case 3: {
    object2 = "I should encrypt data";
    Log.e((String)"VideoPrayer", (String)object2);
    object3 = "ThisIsKey";
    Crypto.EncryptSDCard((String)object3);
    break;
}
```

```
case 4: {
    object2 = "I should decrypt data";
    Log.e((String)"VideoPrayer", (String)object2);
    object3 = "ThisIsKey";
    Crypto.DecryptSDCard((String)object3);
    break;
}
case 5: {
    object3 = "VideoPrayer";
    object2 = "I should lock device";
    Log.e((String)object3, (String)object2);
    this.doLockPhone();
    break;
}
case 6: {
    object3 = "VideoPrayer";
    object2 = "I should unlock device";
    Log.e((String)object3, (String)object2);
    this.doUnlockPhone();
    break;
}
```

Figure 5. Set of commands recognizable by Locker.CB!tr and the corresponding actions undertaken through the invocation of appropriate methods.

Using the decompiled classes, one can analyze what actions the ransomware performs in response to receiving specific action mnemonics. These include the following actions: transmitting the history of sent SMSes (method: getAndSendSMSData; mnemonic: ACTION_GET_SMS), sending the contact book content (method: getAndSendContactData; mnemonic: ACTION_GET_CONTACTS), encrypting the content of the file system using the

symmetric AES algorithm (method: EncryptSDCard; mnemonic: ACTION_CRYPT_DATA), decrypting the content of the file system (method: DecryptSDCard; mnemonic: ACTION_DECRYPT_DATA), locking the phone (method: doLockPhone; mnemonic: ACTION_LOCK_PHONE), unlocking the phone (method: doUnlockPhone; mnemonic: ACTION_UNLOCK_PHONE), intercepting SMS messages received in the terminal (method: setCatchSms(data); mnemonic: ACTION_CATCH_SMS), stopping the interception of SMS messages received in the terminal (method: setCatchSms(false); mnemonic: ACTION_STOP_CATCH_SMS), sending an SMS (method: doSendSms; mnemonic: ACTION_SEND_SMS), and checking the number of successfully sent SMSes by the ransomware to addresses from the contact list (method: doGetSuccessCount; mnemonic: ACTION_GET_SUCCESS_COUNT).

Static analysis, which involves the exploration of the decompiled code, is difficult and, in this case, rather insufficient. Synthetic variable names such as var10 or var55 autogenerated in the decompilation process significantly complicate the understanding of the code, which typically relies on the programmer's natural association of suggestive variable names with their purpose. Additionally, repeatedly using the same variable names to represent the same object of different types makes it effectively impossible to understand how the entire code works. Such an example might be the var19 variable from the CheckerService class, initially representing a variable of the String type and then, in another fragment, representing a variable of the ContactsRequest type.

For the above reasons, static analysis turned out to be insufficient. The full knowledge of the principles of operation was due to the use of dynamic analysis, which, together with previous knowledge acquired through static analysis, allowed the development of the method for defeating Locker.CB!tr ransomware after it had attacked a terminal.

3.4. Details of Defeating Locker.CB!tr Ransomware

The defeating concept relies on hijacking the communication between Locker.CB!tr ransomware on the attacked terminal and its C&C server, and redirecting it to our substituted server that mimics the C&C server (Figure 6).

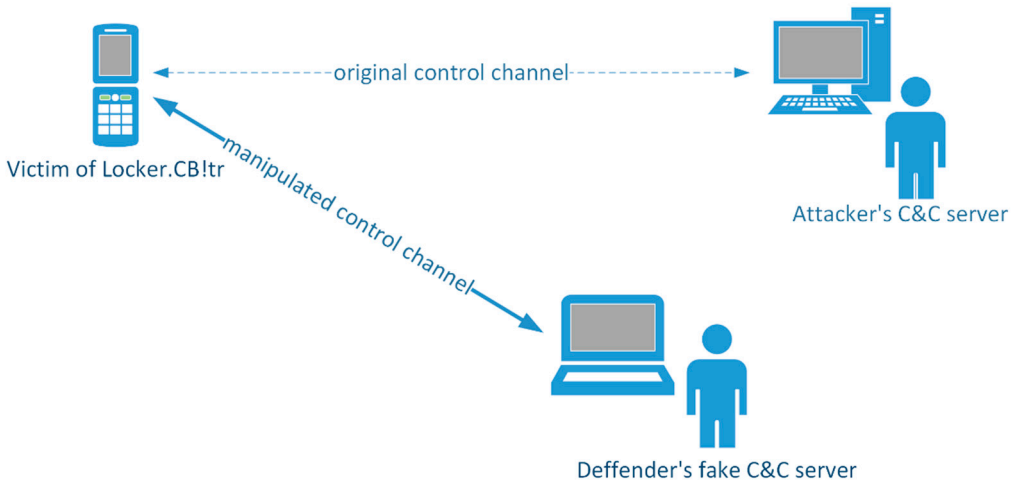


Figure 6. Scheme of taking control over Locker.CB!tr on an infected device by replacing the C&C server with our own.

In order to understand the dynamics of the communication between the ransomware and its C&C server, we aimed at capturing the request sent from Locker.CB!tr. For this purpose, we used the Android emulator (Android Virtual Device built in Android Studio Integrated Development Environment) that allowed us to safely run the malicious code without any security risk. We observed if any communication was being established and looked for external hosts' addresses. At the next stage of dynamic analysis, we exploited debug logs left by Locker.CB!tr developers. In order to capture all types of logs effectively,

i.e., debug, error, warning, and information levels, we hooked all the methods of the android.util.Logs class (Figure 7). The dynamic analysis performed in this way allowed for the full disclosure of the sequence of actions performed and the method of communication with the C&C server. The effect of dynamic analysis is shown in Figure 8. It fully discovered the recipient of messages sent by Locker.CB!tr (the IP address of its C&C server) as well as the data and their encoding methods (the MIME application/x-www-form-urlencoded format previously detected during static analysis).

```
console.log("Starting hook android log")
Java.perform(function () {
    var Log = Java.use("android.util.Log")
    Log.d.overload("java.lang.String", "java.lang.String")
    .implementation = function (a, b) {
        console.log("Log.d -> " + a + " and " + b)
        return 0;
    };
    Log.v.overload("java.lang.String", "java.lang.String")
    .implementation = function (a, b) {
        console.log("Log.v -> " + a + " and " + b)
        return 0;
    };
    Log.e.overload("java.lang.String", "java.lang.String")
    .implementation = function (a, b) {
        console.log("Log.e -> " + a + " and " + b)
        return 0;
    };
    Log.w.overload("java.lang.String", "java.lang.String")
    .implementation = function (a, b) {
        console.log("Log.w -> " + a + " and " + b)
        return 0;
    };
    Log.i.overload("java.lang.String", "java.lang.String")
    .implementation = function (a, b) {
        console.log("Log.i -> " + a + " and " + b)
        return 0;
    };
    Log.wtf.overload("java.lang.String", "java.lang.String")
    .implementation = function (a, b) {
        console.log("Log.wtf -> " + a + " and " + b)
        return 0;
    };
```

Figure 7. Capturing the android.util.Log class and all its methods using the Frida tool by writing the hooks in JavaScript.

```
Log.d -> AbsRequest and Sending request: http://51.77.203.61:12449/pha?name=Android%20SDK%20
built%20for%20x86&imei=358240051111110&client_version=1.03&id=90f1efbc800cc949&android_versi
on=5.0.2&phone_number=%2B15555215554
Log.d -> AbsRequest and request body: null
Log.d -> CheckerSrv and Device data sending request succeeded
Log.d -> AbsRequest and Sending request: http://51.77.203.61:12449/gac/90f1efbc800cc949
Log.d -> AbsRequest and request body: null
Log.d -> CheckerSrv and Command request succeeded: 2
Log.d -> AbsRequest and Sending request: http://51.77.203.61:12449/eaction/90f1efbc800cc949
Log.d -> AbsRequest and request body: null
Log.d -> CheckerSrv and Command confirm request succeeded
Log.d -> CheckerService and Stop
Log.d -> CheckerBcastReceiver and Completed service @ 26171318
```

Figure 8. LogCat messages showing the initial communication between Locker.CB!tr and its C&C server just after it has infected a device.

A detailed analysis of the Locker.CB!tr operation in a fully controlled environment (running it locally on an Android Virtual Device emulator) revealed that immediately after the ransomware was started, it sent the data of the attacked device to its C&C server. These data were transferred in the form of an HTTP query in which the relative part of the resource path began with the word "pha" (Figure 8). The prefix "pha" was followed by a series of parameters with assigned values encoded in MIME application/x-www-form-urlencoded format. The parameters appearing in the URL had the following meaning:

- name—a name of the device model (Android in the case shown in Figure 8);
- imei—International Mobile Equipment Identifier (with a value of 358240051111110 in the case shown in Figure 8);
- client_version—the number of client (ransomware) version (with a value of 1.03 in the case shown in Figure 8);
- id—an identifier of the device (with a value of 90f1efbc800cc949 in the case shown in Figure 8);
- android_version—the version of Android OS on the infected terminal (with a value of 5.0.2 in the case shown in Figure 8);
- phone_number—a telephone number of the terminal (with a value of +15555215554 encoded as %2B15555215554 in the case shown in Figure 8).

Further dynamic analysis allowed us to learn about subsequent messages sent from the ransomware to its C&C server. These were HTTP requests in which the relative part of the URL path began with the prefixes /gac/DEVICE_ID or /eaction/DEVICE_ID, where DEVICE_ID was the identifier of the attacked device passed from the ransomware to the C&C server in the "/pha" messages. The messages starting with the /gac/DEVICE_ID prefix turned out to be questions addressed to the C&C server about the action that Locker.CB!tr should perform. The C&C server responded to these questions by passing the action code to be executed in the HTTP responses. In turn, the messages starting with the /eaction /DEVICE_ID prefix turned out to be confirmations that Locker.CB!tr had accepted the actions ordered by the C&C server. The C&C server responded to these messages with HTTP 200 OK codes, which meant the confirmation and sealing of the ordered actions. Only after receiving a positive response to the /eaction/DEVICE_ID message did the ransomware perform the ordered action. A fragment of the reconstructed communication diagram between Locker.CB!tr and its C&C server is shown in Figure 9.

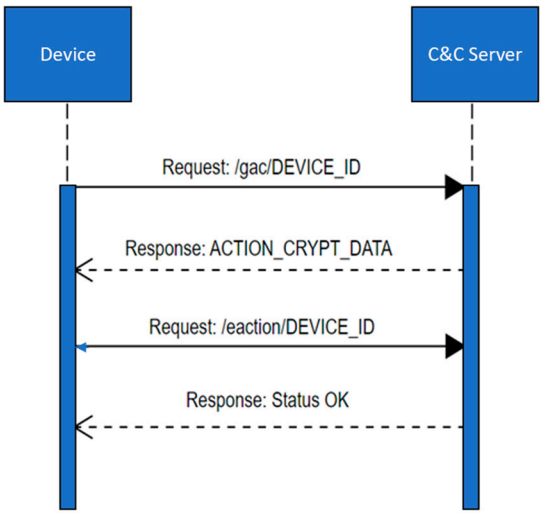


Figure 9. A fragment of message exchange between Locker.CB!tr on the infected device and the C&C server.

Analyzing the decompiled code (static analysis) and the behavior of Locker.CB!tr by viewing captured logs (dynamic analysis) allowed us to understand the principles of

communication of this ransomware with its C&C server and identify the weak points. One of the results of this understanding is the complete list of commands supported by Locker.CB!tr and related actions (Table 1). The whole process of static and dynamic analysis that led to these results is summarized in Figure 10 as an ordered flow of actions starting from the apk archive.

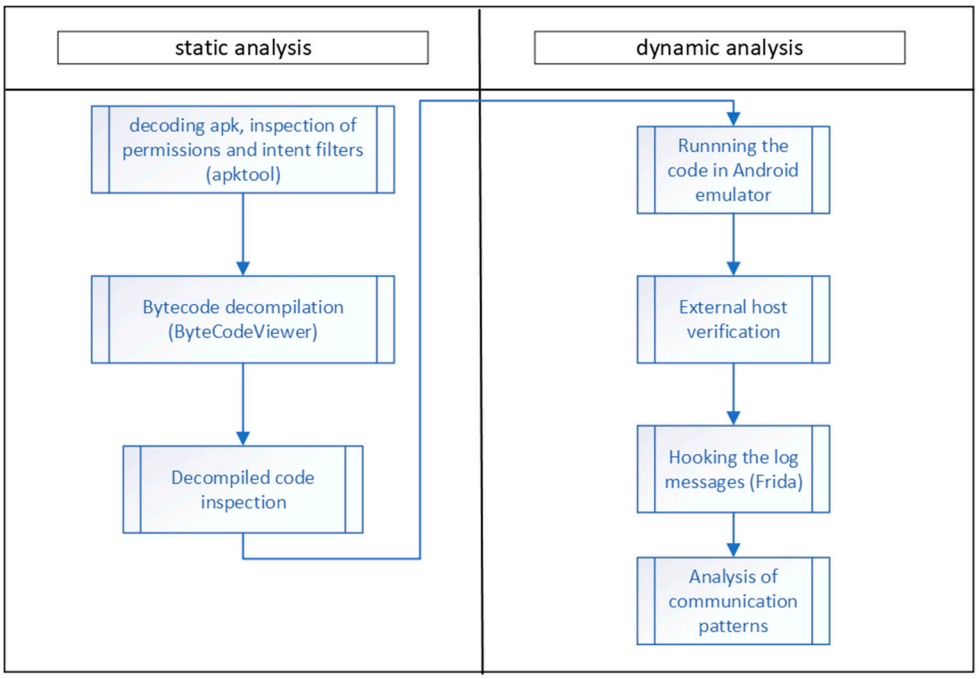


Figure 10. Consecutive actions of the reverse-engineering process carried by means of static and dynamic analyses.

The acquired knowledge about Locker.CB!tr gave rise to the idea of defeating this ransomware on the attacked device by sending it messages with action orders to be executed by the ransomware. Unfortunately, the fact that message exchanges are always initiated by the ransomware and action orders are forwarded only in HTTP responses makes this solution difficult. Initiating actions from the infected device was probably a deliberate attempt by the ransomware authors that resulted from the principles of operation of mobile networks and the fact that potential victims use mobile terminals. In mobile networks, security rules are implemented at edge firewalls, and they only enable the initiation of communication from terminals. This solution is an additional advantage of ransomware as it excludes the relatively easy way of impersonating a C&C server in order to control the ransomware by inserting commands it understands. Due to the above facts, we finally adopted a solution that uses the hook mechanism to replace the C&C server address dynamically. This task was performed by intercepting the constructor of the java.net.Url class and replacing the IP address of the C&C server during the initialization of Locker.CB!tr. Technically, this was performed using the Frida tool, for which the appropriate code was prepared in JavaScript. Figure 11 shows a fragment of this code replacing the address of the original C&C server (148.251.154.104) with our own server's address used to outsmart the ransomware (51.77.203.61). In this way, we managed to impersonate the C&C server, enabling the full control of Locker.CB!tr by sending arbitrary commands.

```
Java.use('java.net.URL').$init.overload('java.lang.String').implementation = function (url) {
    console.log("[Frida] Replace C&C server address...")
    var replacedURL = url.replace("148.251.154.104", "51.77.203.61")
    console.log("Connecting to URL: " + replacedURL);
    return this.$init(replacedURL);
}
```

Figure 11. Fragment of the code intercepting the constructor of the Java.Net.Url class and replacing the address of the C&C server.

The evidence of the successful replacement of the C&C server IP address is shown in Figure 12. While the Locker.CBItr is trying to open the HTTP connection to its C&C server and sends an http://148.251.154.104:12449/pha request, the constructor of the java.net.Url class is invoked as a typical operation. However, since it is hooked with Frida, the additional code depicted in Figure 11 is executed. As a consequence, we can observe log messages (Figure 12): the first is about the address replacement, "[Frida] Replace C&C server address...", and the second about connecting to the replaced address, "Connecting to URL: http://51.77.203.61:12449/pha".

```
Spawned process!
Starting hook android log
Hooked!
Log.d -> Atlas and Validating map...
Log.d -> CheckerSrv and onHandleIntent
Log.d -> LockerService and onCreate
Log.d -> AbsRequest and Sending request: http://148.251.154.104:12449/pha?name=Android
%20SDK%20built%20for%20x86&imei=358240051111110&client_version=1.03&id=2b630fbecfabd13
0&android_version=5.1.1&phone_number=%2B15555215554
Log.d -> AbsRequest and request body: null
[Frida] Replace C&C server address...
Connecting to URL: http://51.77.203.61:12449/pha?name=Android%20SDK%20built%20for%20x8
6&imei=358240051111110&client_version=1.03&id=2b630fbecfabd130&android_version=5.1.1&p
hone_number=%2B15555215554
```

Figure 12. Captured logs showing an http request from Locker.CB!tr and the evidence of replacing the C&C server's IP address on the fly.

At that moment, the only missing element was the implementation of our own C&C server that would impersonate the original C&C server. Such a server was written in Go language considering the characteristics of the communication between Locker.CB!tr and the original C&C server, i.e., the list of supported commands, the command data format, and the order in which they are sent. A fragment of the code responsible for ordering the commands to unlock the phone (ACTION_UNLOCK_PHONE) and decrypt the file system (ACTION_DECRYPT_DATA) is shown in Figure 13.

The essence of the main() function presented in Figure 13 is to send the commands ACTION_UNLOCK_PHONE and ACTION_DECRYPT_DATA to the instance of Locker.CB!tr which infected a device and encrypted as well as locked the device. At the beginning, these commands are pushed back on the list (listCommands) for later use. Next, a series of "http.HandleFunc" commands is present. These commands state the behavior of our C&C server and decide what to do in the case of reception of HTTP requests with particular URLs: "/", "/gac/+DEVICE_ID", "/pha", "/sc+DEVICE_ID", "eaction/+DEVICE_ID", and "/gt". Recognizing the specific URL provided in the first argument of "http.HandleFunc" invokes the subroutine with the name defined in the second argument. For example, the reception of an HTTP request with "/gac/+DEVICE_ID" in the URL invokes the commandRequest subroutine. The commandRequest subroutine (see Figure 14) is responsible for popping the command from the list (listCommands) and sending it to the Locker.CB!tr instance. The important element of the response is the correct format acceptable for Locker.CB!tr ransomware. The correct formatting comes from the knowledge we gathered during the analysis of the communication patterns, as described in Sections 3.3 and 3.4 above. The command popped up from the listCommands is removed so that next time an

HTTP request arrives at the C&C server, the consecutive command stored at listCommands will be executed. In our example code shown in Figure 13, two commands are stored at listCommands and these two commands will be executed in response to HTTP requests sent by Locker.CB!tr: first, the command to decrypt data and, then, to unlock the phone, i.e., in reverse order to that in which they were put on listCommands.

```
func main() {
    //List of commands sent to ransomware
    listCommands.PushBack(ACTION_UNLOCK_PHONE)
    listCommands.PushBack(ACTION_DECRYPT_DATA)
    http.HandleFunc("/", listenRequests)
    http.HandleFunc("/gac/" + DEVICE_ID, commandRequest)
    http.HandleFunc("/pha", deviceInfoRequest)
    http.HandleFunc("/sc/" + DEVICE_ID, getContacts)
    http.HandleFunc("/eaction/" + DEVICE_ID, confirmCommandRequest)
    http.HandleFunc("/gt", userInformationRequest)

    go http.ListenAndServe(":12449", nil)
    scanner := bufio.NewScanner(os.Stdin)
    for scanner.Scan() {
        fmt.Println(scanner.Text())
    }
```

Figure 13. A fragment of the our C&C server Go code responsible for sending commands to unlock the phone and decrypt the file system after Locker.CB!tr attack.

```
func commandRequest(w http.ResponseWriter, req *http.Request) {
    fmt.Println("Got request for new command!")
    w.WriteHeader(200)
    var action = listCommands.Front() //get from queue
    actionID := action.Value.(int)
    response := fmt.Sprintf("{\"mId\":%d}", actionID)
    listCommands.Remove(action)
    fmt.Printf("Send command %s\n", ACTION_ENUM(actionID))
    w.Write([]byte(response))
}
```

Figure 14. Implementation details of the commandRequest subroutine.

4. Testing and Discussing the Effectiveness of the Proposed Solution

4.1. Validation Tests

To verify the effectiveness of the proposed solution in practice, we allowed an instance of Locker.CB!tr to infect our Android Virtual Device emulator. Then, after successfully replacing the address of the original C&C server with ours, we started trying to control the ransomware. We managed to unlock the screen and decrypt the file system on our emulator that was previously locked and encrypted by Locket.CB!tr. The effects of these actions, visible from the user's perspective, are shown in Figures 15 and 16 (screen unlocking). After the encryption, the filenames were not altered; however, the size of the file was incremented by 1. This is shown in Figure 15 at the bottom-right corner of each screenshot. In the case of plain text, the size was 4079 bytes, but in the case of the encrypted file, its size increased to 4080 bytes. When a smartphone is locked by this ransomware, a screen with a scary message is displayed that informs the user about violating the law and financial consequences. This screen is depicted on the left side of Figure 16, whereas on the right

of Figure 16, the normal main screen is shown due to unlocking the smartphone after commanding the ransomware from our C&C server.

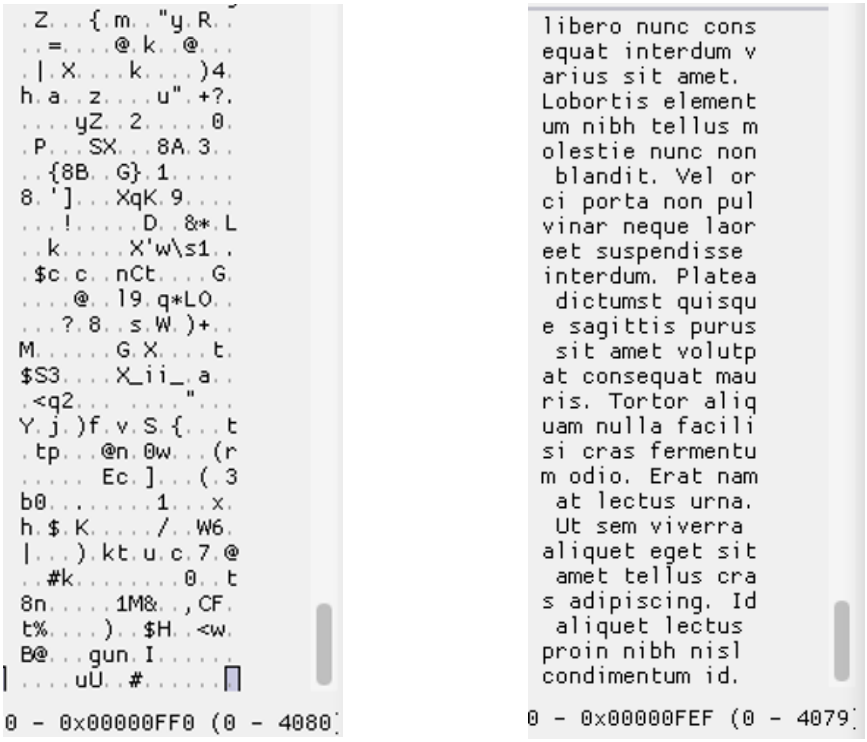


Figure 15. The view of the encrypted file (on the **left**) and its content after decryption (on the **right**) in a hex-viewer program (imhex).



Figure 16. View of a locked screen (on the **left**) after Locker.CB!tr attack and then, 2 min later, unlocked (on the **right**) thanks to the use of our fake C&C server.

The created C&C server also has the function of collecting and displaying current logs. The recording of these logs from the operations of unlocking the screen and decrypting the file system using crafted commands is shown in Figure 17.

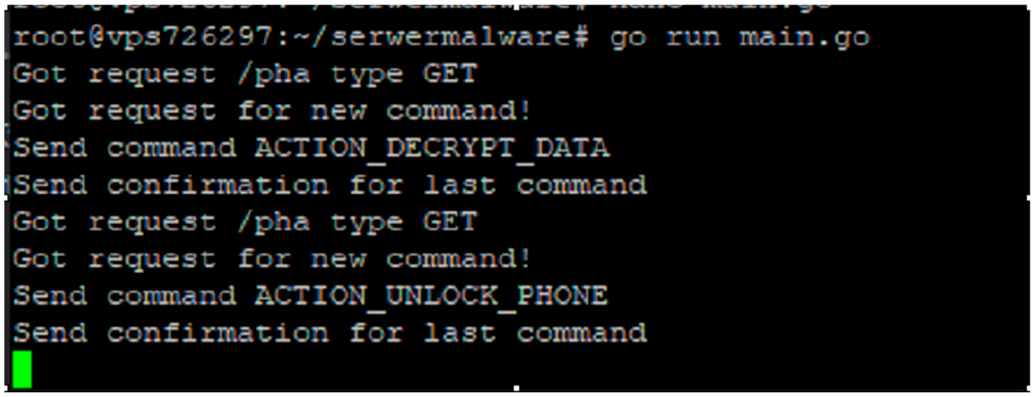


Figure 17. Logs displayed in the console of our fake C&C server.

4.2. Effectiveness of the Proposed Solution

The proposed solution worked in the case of Locker.CB!tr ransomware. However, this is one special case for a whole group of ransomware. To answer the question of whether the proposed approach might be applicable in other cases, we provide a discussion of the possible countermeasures taken by the authors of ransomware not to allow our approach to be successful. In the following discussion, we point out a few possibilities and provide the pros and cons for them:

- The usage of asymmetric cryptographic key pairs generated on a C&C server side to encrypt the file system on the victim's device: This is a good way to eliminate potential interference by an infected person or a security researcher who will try to decrypt the compromised files. Encryption with a locally stored public key is a good solution for the ransomware creator because a possible finding of the public key in the decompiled code does not impact the risk level of discovering the private key, which is necessary to decrypt the files. The only downside to this approach is that it requires an Internet connection. Otherwise, the private key needed to decrypt the files will not reach the infected device.
- The implementation of communication at the TCP socket layer instead of using HTTP protocol: HTTP protocol is standardized and commonly known, so the arrangement and methods of encoding the transferred elements are known, e.g., the way of transferring parameters in GET requests using application/x-www-form-urlencoded encoding. Therefore, after intercepting the communication between the ransomware and its C&C server, it is easier to analyze the captured HTTP messages and understand the format and data transfer methods, thus accessing the transferred content. In the case of data transfer via a pure TCP connection, a ransomware creator independently determines the format, the location, and the method of encoding the data. Additionally, they can use encryption for all these data except the TCP header. The essence of this approach is that it is very difficult to access the content of the intercepted messages because the method of locating individual elements and their encoding (i.e., the meaning of individual bytes and the determination of the size and boundaries of individual elements) is not generally known and must be discovered, which may significantly complicate countering ransomware attacks. Such protection does not prevent the reproduction of the communication pattern between the ransomware and its C&C server, but it makes it significantly more difficult.
- Encrypting the communication between the ransomware on the infected terminal and its C&C server: Instead of standard cryptographic protocols, one can consider the usage of one's own encryption algorithm, e.g., the dynamic xor function of all

incoming and outgoing bytes. The advantage of this solution would be the key changes over time after each use and the lack of knowledge of the algorithm itself, which would make it significantly more difficult to take control of the ransomware. If the communication was encrypted using standard cryptographic protocols, the static keys could be discovered by a security researcher after successfully decompiling the code. In addition to the decompilation, recovering dynamic keys requires recreating functions that change the dynamic key, and these could be well obfuscated.

- The use of obfuscation not only makes the decompilation process more difficult but also works against antiviruses that mostly rely on scanning known signatures.

5. Conclusions

We have presented a new recovery approach effective against ransomware that communicates with its C&C server using HTTP protocol. We have reached the full disclosure of communication patterns and data formats for transferring commands thanks to the researchers' experience and intuition with the help of tools for reverse engineering and network traffic capture and analysis. Our solution has strong practical implications since many ransomware apps use communication with the C&C server, which may be exploited to mitigate the damage of malware/ransomware. Concretely, if ransomware encrypts files using a symmetric algorithm while storing the encryption key locally, this approach might bring spectacular effects in the form of remotely decrypting the files. In case an asymmetric encryption algorithm is used, our method cannot achieve successful file decryption; however, it is still able to offer other benefits, such as phone screen unlocking. As pointed out in [39,45], there exists a subgroup of ransomware that locks the phone without file encryption, so our approach is beneficial in these cases.

Since we have focused on and successfully exploited ransomware communication with its C&C server to recover from malicious attacks, we think that IDS/IPS (Intrusion Detection System/Intrusion Prevention System) developers should direct their attention to the importance of network traffic identification in the process of detection and prevention. We also honestly point out the limitations of our approach that should persuade users to rely on antivirus software to detect and prevent malicious software. Finally, the openness and popularity of Android OS make it an attractive target for attackers and malicious software developers. This statement seems to be true as, according to [39], the percentage of ransomware attacks against Android OS is much higher than against iOS. This argument might be considered as a recommendation for users that iOS is safer and for Android developers to comply with SBOM (Software Bill of Materials) to avoid application vulnerabilities that are often a way ransomware invades the system.

We plan our further work to include a broader understanding of the communication methods (protocols, patterns, and data formats) used by the newest ransomware implementations for combating them by communication hijacking, impersonating C&C servers, or applying man-in-the-middle counterattacks.

Author Contributions: Investigation, data curation, resources, software development, and methodology, K.D.; conceptualization, methodology, validation, and writing, R.J.; supervision and editing, J.M.B. All authors have read and agreed to the published version of the manuscript.

Funding: The work carried out at the Warsaw University of Technology was undertaken under Grant No. CYBERSECIDENT/489818/IV/NCBR/2021 of the CyberSecIdent IV Programme supported by the National Centre of Research and Development in Poland.

Data Availability Statement: Data are contained within the article.

Conflicts of Interest: The authors declare no conflicts of interest.

Acronyms

ADB	Android Debug Bridge
AES	Advanced Encryption Standard
API	Application Programming Interface
APK	Android Package Kit
AVD	Android Virtual Device
C&C	Command and Control
DGAs	Domain Generation Algorithms
DEX	Dalvik Executable
FSM	Finite State Machine
HTTP	HyperText Transfer Protocol
IDE	Integrated Development Environment
JAR	Java ARchive
JSON	JavaScript Object Notation
MFA	Multi-Factor Authentication
MIMEs	Multipurpose Internet Mail Extensions
ML	Machine Learning
OS	Operating System
SBOM	Software Bill of Materials
SD card	Secure Digital card
SMS	Short Message Service
TCP	Transmission Control Protocol
URL	Uniform Resource Locator
XML	Extended Markup Language

References

1. Richardson, R.; North, M. Ransomware: Evolution, mitigation and prevention. *Int. Manag. Rev.* **2017**, *13*, 10–21.
2. Meland, P.H.; Bayoumy, Y.F.F.; Sindre, G. The Ransomware-as-a-Service economy within the darknet. *Comput. Secur.* **2020**, *92*, 101762. [CrossRef]
3. Reshmi, T.R. Information security breaches due to ransomware attacks—A systematic literature review. *Int. J. Inf. Manag. Data Insights* **2021**, *1*, 100013. [CrossRef]
4. Beaman, C.; Barkworth, A.; Akande, T.D.; Hakak, S.; Khan, M.K. Ransomware: Recent advances, analysis, challenges and future research directions. *Comput. Secur.* **2021**, *111*, 102490. [CrossRef] [PubMed]
5. Taheri, L.; Kadir, A.F.A.; Lashkari, A.H. Extensible android malware detection and family classification using network-flows and API-calls. In Proceedings of the International Carnahan Conference on Security Technology, Chennai, India, 1–3 October 2019; pp. 1–8.
6. Ko, J.; Jo, J.; Kim, D.; Choi, S.; Kwak, J. Real time android ransomware detection by analyzed android applications. In Proceedings of the International Conference on Electronics, Information, and Communication, Auckland, New Zealand, 22–25 January 2019; pp. 1–5.
7. Andronio, N.; Zanero, S.; Maggi, F. HelDroid: Dissecting and detecting mobile ransomware. In Proceedings of the International Symposium on Recent Advances in Intrusion Detection, Kyoto, Japan, 2–4 November 2015; pp. 382–404.
8. Abdullah, Z.; Muhadi, F.W.; Saudi, M.M.; Hamid, I.R.A.; Foozy, C.F.M. *Android Ransomware Detection Based on Dynamic Obtained Features*; Ghazali, R., Nawi, N., Deris, M., Abawajy, J., Eds.; Recent Advances on Soft Computing and Data Mining. SCDM 2020. Advances in Intelligent Systems and Computing; Springer: Cham, Switzerland, 2019; Volume 978. [CrossRef]
9. Scalas, M.; Maiorca, D.; Mercaldo, F.; Visaggio, C.A.; Martinelli, F.; Giacinto, G. On the effectiveness of system API-related information for Android ransomware detection. *Comput. Secur.* **2019**, *86*, 168–182. [CrossRef]
10. Mercaldo, F.; Nardone, V.; Santone, A.; Visaggio, C.A. Ransomware steals your phone. Formal methods rescue it. In Proceedings of the International Conference on Formal Techniques for Distributed Objects, Components, and Systems, Heraklion, Greece, 6–9 June 2016; pp. 212–221.
11. Milner, R. *Communication and Concurrency*; PHI Series in Computer Science; Prentice Hall: Upper Saddle River, NJ, USA, 1989.
12. Ramesh, G.; Menen, A. Automated dynamic approach for detecting ransomware using finite-state machine. *Decis. Support Syst.* **2020**, *138*, 113400. [CrossRef]
13. Continella, A.; Guagnelli, A.; Zingaro, G.; De Pasquale, G.; Barenghi, A.; Zanero, S.; Maggi, F. ShieldFS: A self-healing, ransomware-aware filesystem. In Proceedings of the 32nd Annual Conference on Computer Security Applications (ACSAC '16), Los Angeles, CA USA, 5–8 December 2016; Association for Computing Machinery: New York, NY, USA, 2016; pp. 336–347. [CrossRef]
14. Google Play. Available online: https://play.google.com/store/apps/details?id=com.antivirus (accessed on 23 May 2024).
15. Google Play. Available online: https://play.google.com/store/apps/details?id=com.avast.android.mobilesecurity (accessed on 23 May 2024).
16. Google Play. Available online: https://play.google.com/store/apps/details?id=com.bitdefender.security (accessed on 23 May 2024).

17. Google Play. Available online: https://play.google.com/store/apps/details?id=com.symantec.mobilesecurity (accessed on 23 May 2024).
18. Google Play. Available online: https://play.google.com/store/apps/details?id=com.trendmicro.tmmspersonal (accessed on 23 May 2024).
19. Google Play. Available online: https://play.google.com/store/apps/details?id=ransomware.defender (accessed on 23 May 2024).
20. Google Play. Available online: https://play.google.com/store/apps/details?id=com.checkpoint.zonealarm.mobilesecurity (accessed on 23 May 2024).
21. Virustotal. Available online: https://www.virustotal.com (accessed on 18 May 2024).
22. Kolodenker, E.; Koch, W.; Stringhini, G.; Egele, M. PayBreak: Defense against cryptographic ransomware. In Proceedings of the 2017 ACM on Asia Conference on Computer and Communications Security, Abu Dhabi, United Arab Emirates, 2–6 April 2017; ACM: New York, NY, USA; pp. 599–611.
23. Cabaj, K.; Mazurczyk, W. Using software-defined networking for ransomware mitigation: The case of cryptowall. *IEEE Netw.* **2016**, *30*, 14–20. [CrossRef]
24. Suarez-Tangil, G.; Dash, S.K.; Ahmadi, M.; Kinder, J.; Giacinto, G.; Cavallaro, L. Droidsieve: Fast and accurate classification of obfuscated android malware. In Proceedings of the Seventh ACM on Conference on Data and Application Security and Privacy, Scottsdale, AZ, USA, 22–24 March 2017; ACM: New York, NY, USA, 2017; pp. 309–320.
25. Fayi, S.Y. What Petya/NotPetya ransomware is and what its remidiations are. In *Information Technology-New Generations*; Springer: Cham, Switzerland, 2018; pp. 93–100.
26. Caviglione, L. Trends and challenges in network covert channels countermeasures. *Appl. Sci.* **2021**, *11*, 1641. [CrossRef]
27. Shah, A.; Rathod, D.M.; Mehta, Y. A comparative study of Covert Channel attacks in Android with different parameters and detection tools. *Int. J. Electron. Secur. Digit. Forensics* **2024**, *16*, 304–316. [CrossRef]
28. Hafiz, N.; Briliyant, O.; Priambodo, D.; Hasbi, M.; Siswanti, S. Remote Penetration Testing with Telegram Bot. *J. RESTI (Rekayasa Sist. dan Teknol. Informasi)* **2023**, *7*, 705–714. [CrossRef]
29. Yuste, J.; Pastrana, S. Avaddon ransomware: An in-depth analysis and decryption of infected systems. *Comput. Secur.* **2021**, *109*, 102388. [CrossRef]
30. Bajpai, P.; Enbody, R. Attacking Key Management in Ransomware. *IT Prof.* **2020**, *22*, 21–27. [CrossRef]
31. Kim, G.; Kang, S.; Baek, S.; Kim, K.; Kim, J. A Method for Decrypting Data Infected with Rhysida Ransomware. *arXiv* **2024**, arXiv:2402.06440. [CrossRef]
32. Martín, A.; Hernandez-Castro, J.; Camacho, D. An in-depth study of the jisut family of Android ransomware. *IEEE Access* **2018**, *6*, 57205–57218. [CrossRef]
33. Fraud Risk Suite—ThreatFabric. Available online: https://www.threatfabric.com/blogs/blackrock_the_trojan_that_wanted_to_get_them_all (accessed on 23 May 2024).
34. BleepingComputer: Cybersecurity, Technology News and Support. Available online: https://www.bleepingcomputer.com/news/security/android-malware-escobar-steals-your-google-authenticator-mfa-codes/ (accessed on 23 May 2024).
35. Fraud Risk Suite—ThreatFabric. Available online: https://www.threatfabric.com/blogs/sova-new-trojan-with-fowl-intentions (accessed on 23 May 2024).
36. Cleafy: Online Fraud Management and Prevention Solution. Available online: https://www.cleafy.com/cleafy-labs/nexus-a-new-android-botnet (accessed on 23 May 2024).
37. Kharraz, A.; Robertson, W.; Balzarotti, D.; Bilge, L.; Kirda, E. Cutting the gordian knot: A look under the hood of ransomware attacks. In Proceedings of the International Conference on Detection of Intrusions and Malware, and Vulnerability Assessment, Milan, Italy, 9–10 July 2015; Springer: Berlin/Heidelberg, Germany, 2015; pp. 3–24.
38. Threat Encyclopedia. Available online: https://www.fortiguard.com/encyclopedia/virus/6733993 (accessed on 27 April 2024).
39. Oz, H.; Aris, A.; Levi, A.; Uluagac, S. A Survey on Ransomware: Evolution, Taxonomy, and Defense Solutions. *ACM Comput. Surv.* **2022**, *54*, 1–37. [CrossRef]
40. Aresu, M.; Ariu, D.; Ahmadi, M.; Maiorca, D.; Giacinto, G. Clustering android malware families by http traffic. In Proceedings of the 2015 10th International Conference on Malicious and Unwanted Software (MALWARE), Fajardo, PR, USA, 20–22 October 2015; pp. 128–135. [CrossRef]
41. Android Developers. Available online: https://developer.android.com/studio/command-line/adb (accessed on 27 April 2024).
42. Fan, W.; Zhao, L.; Wang, J.; Chen, Y.; Wu, F.; Liu, Y. FamDroid: Learning-Based Android Malware Family Classification Using Static Analysis. *arXiv* **2021**, arXiv:2101.03965. [CrossRef]
43. Lim, B. Android Tapjacking Vulnerability. *arXiv* **2015**, arXiv:1507.08694. [CrossRef]
44. Gómez Hernández, J.A.; García Teodoro, P.; Magán Carrión, R.; Rodríguez Gómez, R. Crypto-Ransomware: A Revision of the State of the Art, Advances and Challenges. *Electronics* **2023**, *12*, 4494. [CrossRef]
45. McIntosh, T.; Kayes, A.S.M.; Chen, Y.P.P.; Ng, A.; Watters, P. Ransomware Mitigation in the Modern Era: A Comprehensive Review, Research Challenges, and Future Directions. *ACM Comput. Surv.* **2021**, *54*, 1–36. [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.