Enhancing Security and Performance through Customized Android ROMs

Chetan Reddy Bojja, CS19B012

Research Guide : Dr. Chester Rebeiro

Introduction

- Android OS, known for its open-source nature and customizability, is widely used not just in smartphones but also in single board computers (SBCs), IoT devices, and server systems.

- The vast user base of Android presents remarkable opportunities for developers and researchers, but it also brings significant challenges, especially in the realms of security and performance

- The project aims to enhance security and performance of Android-based devices by customizing Android ROMs, focusing on the reduction of unnecessary applications or bloatware.

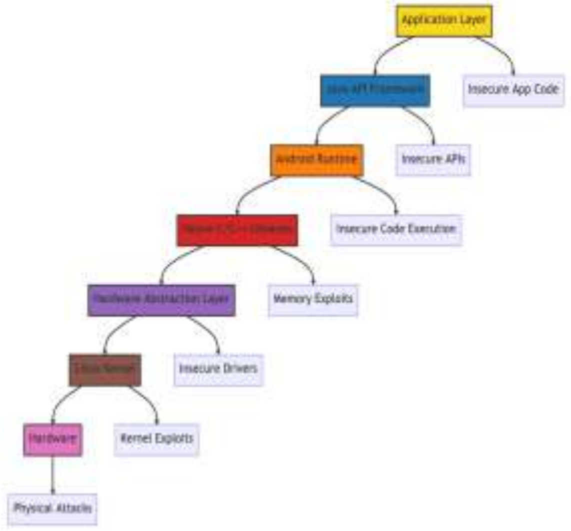
- Our approach is shaped by Ross Anderson's insight stating, "A system with a minimal function set is easier to secure, while feature-rich systems are usually complex and hence hard to protect."

Vulnerabilities in Android Systems

- **Potential Security Threats:** Android's widespread use and open-source nature make it a common target for cyberattacks. These threats can range from sophisticated malware to data leakage due to insecure app permissions.

- **Privacy Concerns:** Some pre-installed applications (bloatware) can have excessive permissions, leading to potential privacy concerns. For instance, a pre-installed app might have access to sensitive data it doesn't need for its function.

- **Real-life example - 'xHelper' Malware:** First discovered in 2019, the 'xHelper' malware, pre-installed on some Android devices, underscored the significant security risks associated with bloatware. Despite manual removal and factory resets, this persistent malware continued to reappear, emphasizing the need for effective security measures in Android systems.



Potential Solutions to Android Vulnerabilities

- **Building an OS from Scratch:** While this option offers the most control over the OS, it requires significant resources and expertise in OS development and maintenance.

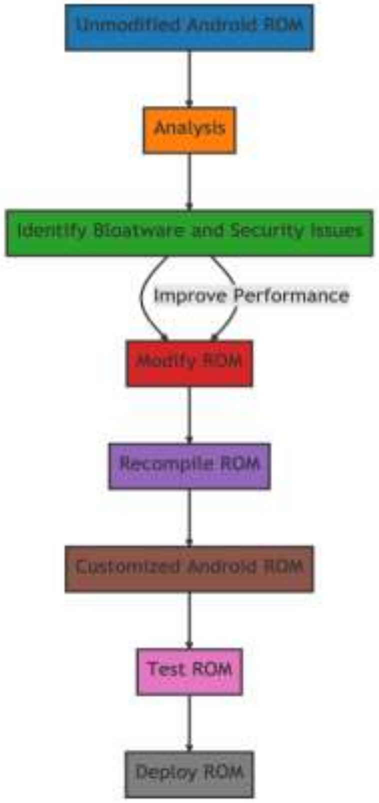
- **Minimalist Operating Systems and Microkernel Architecture:** These OSs aim to minimize the attack surface by reducing the amount of code running in the privileged mode. However, they may not support all the functionalities provided by Android.

- **Customizing Existing Android OS (Android ROMs):** This method involves removing unnecessary applications and integrating desired applications into the Android Open Source Project (AOSP). It's more practical and efficient compared to the other options, as it leverages existing resources and knowledge base.

- These solutions each have their pros and cons, but in the context of this project, we found customizing the Android ROMs to be the most effective approach.

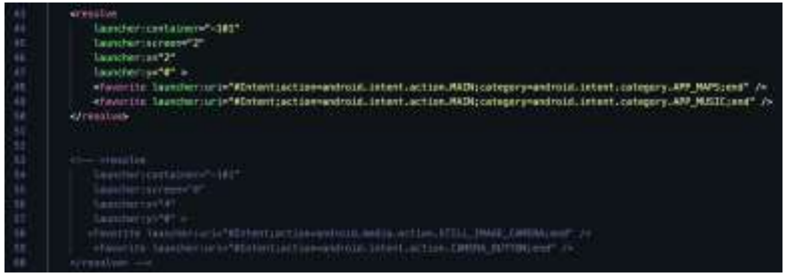
Project Implementation

- The project involves customizing an Android ROM for ARM-based Tinker Board S R2.0 SBC, primarily targeting performance and security enhancements.
- The custom ROM development process includes the removal of unwanted apps (bloatware), integration of custom applications and the optional UI customization.
- Two methods have been employed for launching custom applications - as a background process during system boot or as a foreground process post system boot.

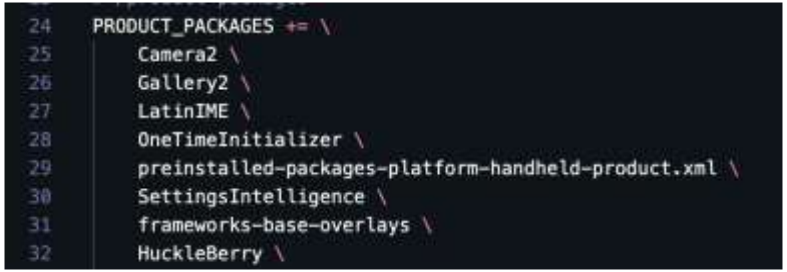


Customization & Removing Bloatware

- **UI Customization:** Located the system files responsible for wallpapers, icons and modified them to likeness.
- **Bloatware Identification:** Identified the system packages and libraries that were non-essential or rarely used, classifying them as bloatware.
- **Bloatware Removal:** Modified the Android Make files to exclude the identified bloatware, effectively removing them during the build process.



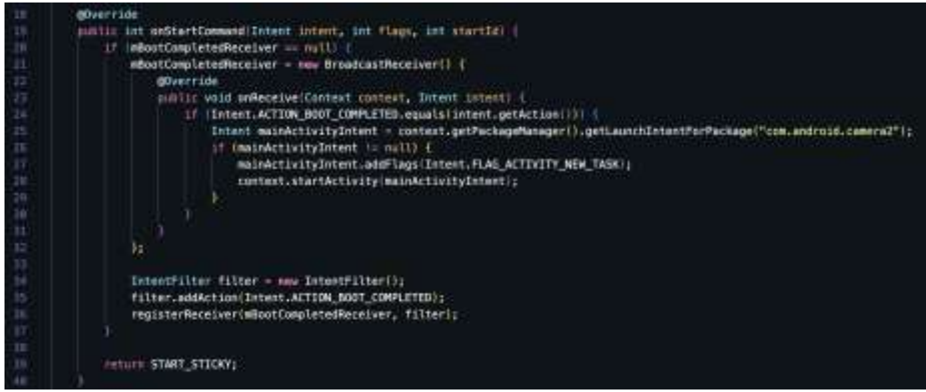
```
<resolve
    launcher:container="-101"
    launcher:screen="2"
    launcher:x="2"
    launcher:y="0" >
    <favorite launcher:uri="#Intent;action=android.intent.action.MAIN;category=android.intent.category.APP_MAPS;end" />
    <favorite launcher:uri="#Intent;action=android.intent.action.MAIN;category=android.intent.category.APP_MUSIC;end" />
</resolve>


<!-- <resolve
    launcher:container="-101"
    launcher:screen="0"
    launcher:x="4"
    launcher:y="0" >
   <favorite launcher:uri="#Intent;action=android.media.action.STILL_IMAGE_CAMERA;end" />
    <favorite launcher:uri="#Intent;action=android.intent.action.CAMERA_BUTTON;end" />
</resolve> -->
```



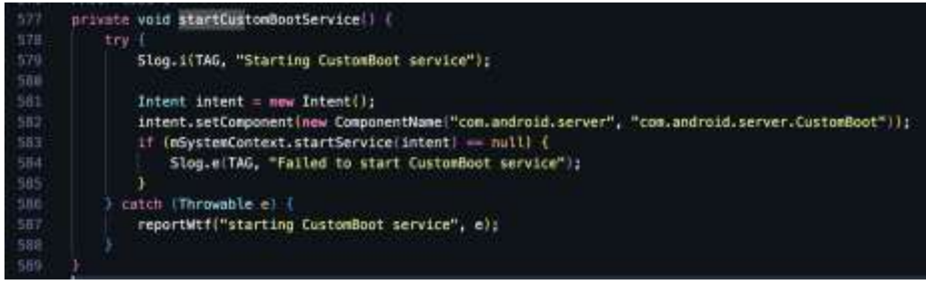
```
PRODUCT_PACKAGES += \
    Camera2 \
    Gallery2 \
    LatinIME \
    OneTimeInitializer \
    preinstalled-packages-platform-handheld-product.xml \
    SettingsIntelligence \
    frameworks-base-overlays \
    HuckleBerry \
```

Background Launching of Apps

- **Integrating custom app into boot sequence:** In this module, I modified core OS files (SystemServer.java & CustomBoot.java to incorporate the custom app into the system boot sequence, enabling it to run as a background process during boot.



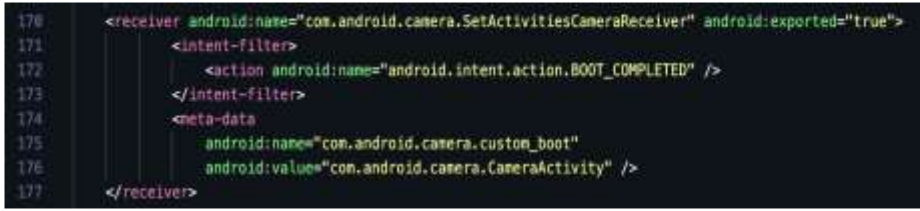
```java
@Override
public int onStartCommand(Intent intent, int flags, int startId) {
    if (mBootCompletedReceiver == null) {
        mBootCompletedReceiver = new BroadcastReceiver() {
            @Override
            public void onReceive(Context context, Intent intent) {
                if (Intent.ACTION_BOOT_COMPLETED.equals(intent.getAction())) {
                    Intent mainActivityIntent = context.getPackageManager().getLaunchIntentForPackage("com.android.camera2");
                    if (mainActivityIntent != null) {
                        mainActivityIntent.addFlags(Intent.FLAG_ACTIVITY_NEW_TASK);
                        context.startActivity(mainActivityIntent);
                    }
                }
            }
        };

        IntentFilter filter = new IntentFilter();
        filter.addAction(Intent.ACTION_BOOT_COMPLETED);
        registerReceiver(mBootCompletedReceiver, filter);
    }

    return START_STICKY;
}
```



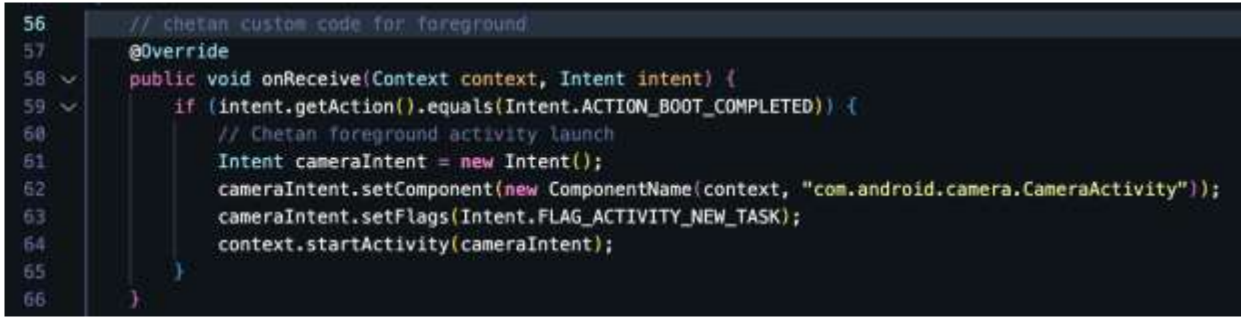
```java
private void startCustomBootService() {
    try {
        Slog.i(TAG, "Starting CustomBoot service");

        Intent intent = new Intent();
        intent.setComponent(new ComponentName("com.android.server", "com.android.server.CustomBoot"));
        if (mSystemContext.startService(intent) == null) {
            Slog.e(TAG, "Failed to start CustomBoot service");
        }
    } catch (Throwable e) {
        reportWtf("starting CustomBoot service", e);
    }
}
```

Foreground Launching of Apps

- **Modifying app manifest and adding receiver code:** The foreground method involved altering the custom app's manifest and adding specific receiver code. This allowed the app to listen for the BOOT_COMPLETED broadcast intent and start up as a foreground process after boot.



```
<receiver android:name="com.android.camera.SetActivitiesCameraReceiver" android:exported="true">
    <intent-filter>
        <action android:name="android.intent.action.BOOT_COMPLETED" />
    </intent-filter>
    <meta-data
        android:name="com.android.camera.custom_boot"
        android:value="com.android.camera.CameraActivity" />
</receiver>
```



```
// chetan custom code for foreground
@Override
public void onReceive(Context context, Intent intent) {
    if (intent.getAction().equals(Intent.ACTION_BOOT_COMPLETED)) {
        // Chetan foreground activity launch
        Intent cameraIntent = new Intent();
        cameraIntent.setComponent(new ComponentName(context, "com.android.camera.CameraActivity"));
        cameraIntent.setFlags(Intent.FLAG_ACTIVITY_NEW_TASK);
        context.startActivity(cameraIntent);
    }
}
```

Background vs Foreground Comparison

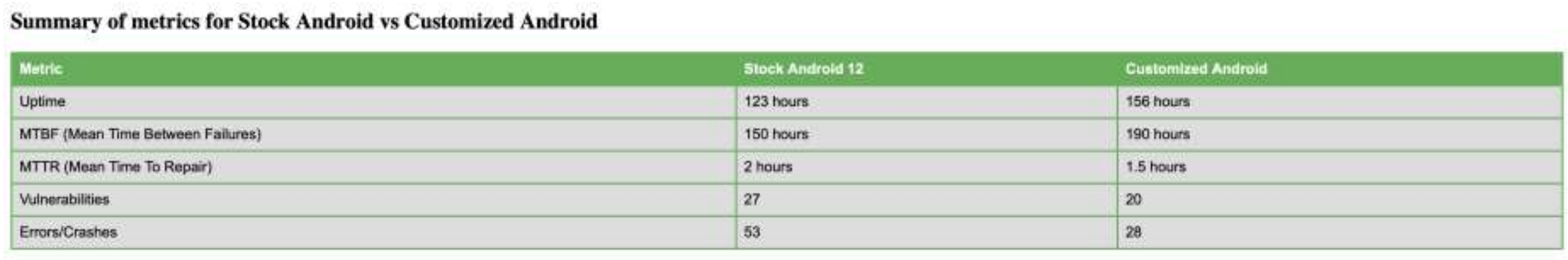
- **Load times:** The background method offers faster load times, while the foreground method may face performance penalties if multiple apps are started.

- **Security concerns:** Background method might pose security risks due to root privileges, while foreground method provides user-level privilege, offering more security after boot.

- **Virus/Malware Threat:** The background method could enable virus/malware running at boot time, while the foreground method does not have this issue.

- **Choice of method:** The choice between the two depends on the specific needs and trade-offs between performance and security.

Experimental Evaluation



Summary of metrics for Stock Android vs Customized Android

Metric	Stock Android 12	Customized Android
Uptime	123 hours	156 hours
MTBF (Mean Time Between Failures)	150 hours	190 hours
MTTR (Mean Time To Repair)	2 hours	1.5 hours
Vulnerabilities	27	20
Errors/Crashes	53	28

Applicability to Wider Range of Devices

- **Android Platform Universality:** The Android platform's versatility and widespread usage across devices provide a broad application base for our research. Our techniques are not confined to the Tinker Board SBC or a specific Android version.

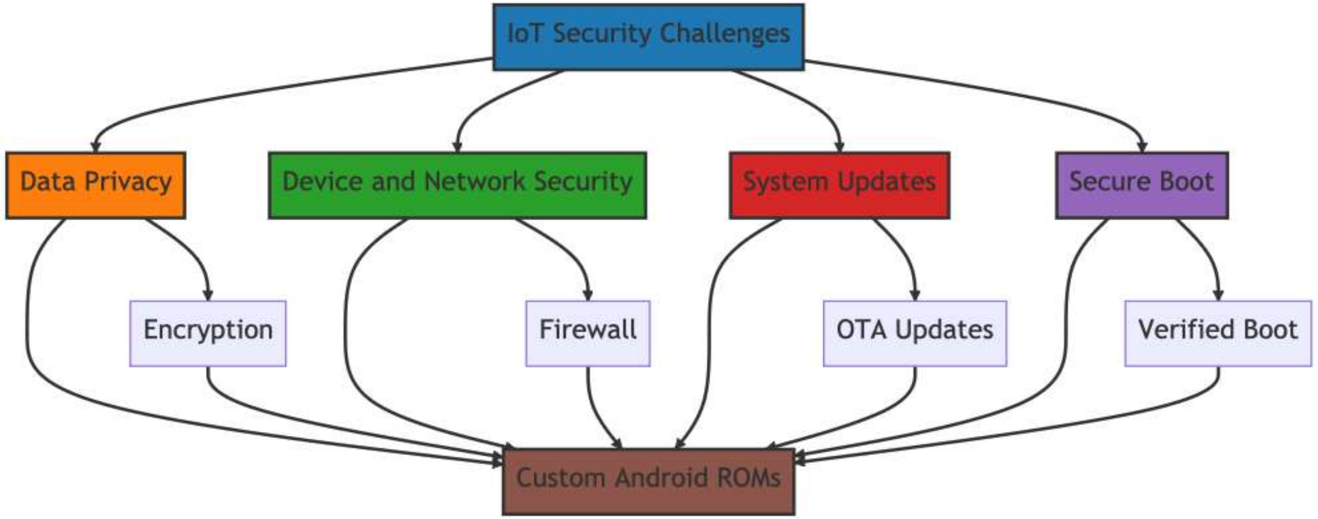
- **Customization Flexibility:** The customization principles applied in our project can be adapted to optimize Android ROMs on a wide array of devices. This includes smartphones, IoT devices, and other SBCs.

- **Security Enhancement:** The security measures we applied, like minimizing unnecessary features and complexity, are universally beneficial for enhancing the overall security of any device running on Android.

- **Performance Improvement:** Similar performance improvements, such as the removal of bloatware and efficient app launching, can be beneficial for different Android devices and versions.

Applicability to Wider Range of Devices

- The project's customized, streamlined, and secure Android ROM holds potential for broad applicability, enhancing performance and security across a variety of domains including Internet of Things (IoT) devices, enterprise applications, data-sensitive industries, older devices, car infotainment systems, privacy-focused users, gaming and VR devices, media servers, multimedia devices, and blockchain nodes.



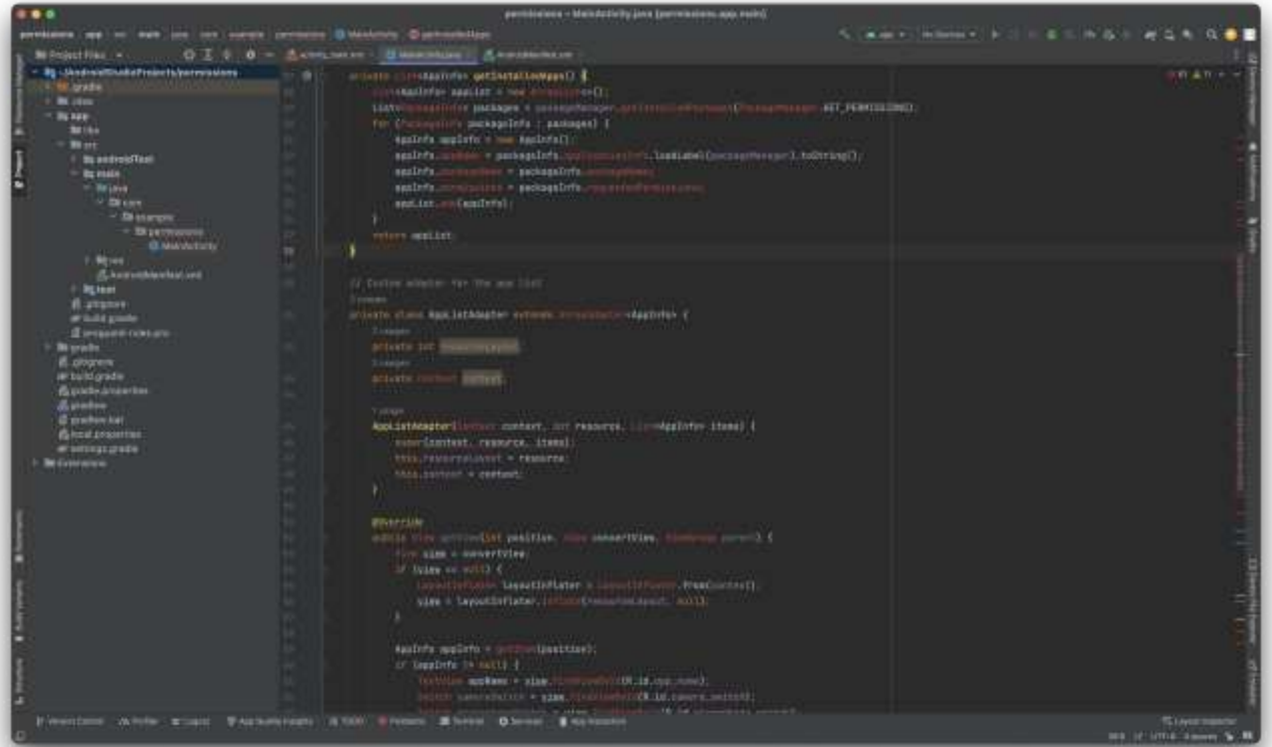
Future Work

- As the project evolves, my current focus is on developing an Android application designed to launch as a background process and manage permissions for all apps on the device. This application will serve to bolster security features even further by adding an additional layer of control over app permissions.

- Currently, the application's development is about 70-80% complete, with the bulk of the coding phase nearing its completion. The remaining work primarily involves refining the code, testing its functionality, and ensuring seamless integration with the customized Android ROM.

- My goal is to provide an intuitive and powerful tool that will further enhance the security and control users have over their devices, pushing the boundaries of what our customized Android ROM can offer.

Future Work



```java
private List<AppInfo> getInstalledApps() {
    List<AppInfo> appList = new ArrayList<>();
    List<PackageInfo> packages = packageManager.getInstalledPackages(PackageManager.GET_PERMISSIONS);
    for (PackageInfo packageInfo : packages) {
        AppInfo appInfo = new AppInfo();
        appInfo.appName = packageInfo.applicationInfo.loadLabel(packageManager).toString();
        appInfo.packageName = packageInfo.packageName;
        appInfo.permissions = packageInfo.requestedPermissions;
        appList.add(appInfo);
    }
    return appList;
}

// Custom adapter for the app list
private class AppListAdapter extends ArrayAdapter<AppInfo> {
    private int resourceLayout;
    private Context context;

    AppListAdapter(Context context, int resource, List<AppInfo> items) {
        super(context, resource, items);
        this.resourceLayout = resource;
        this.context = context;
    }

    @Override
    public View getView(int position, View convertView, ViewGroup parent) {
        View view = convertView;
        if (view == null) {
            LayoutInflater layoutInflater = LayoutInflater.from(context);
            view = layoutInflater.inflate(resourceLayout, null);
        }

        AppInfo appInfo = getItem(position);
        if (appInfo != null) {
            TextView appName = view.findViewById(R.id.app_name);
            Switch cameraSwitch = view.findViewById(R.id.camera_switch);
```

Conclusions

- **Systematic Approach:** Provided a systematic methodology for analyzing and customizing Android ROMs for heightened security and performance.

- **Performance Optimization:** Streamlined the Android ROM by eliminating unnecessary applications, enhancing overall system performance and reliability.

- **Visual Customization:** Personalized UI elements for a visually appealing user experience, enhancing the user interface as per the user's preferences.

- **Startup Applications:** Showcased detailed guidelines for modifying source code to add custom applications and automatically launch specific applications at startup.

THANK YOU