

Overview of the Sambodana Project: Development of Mobile Communication Security System using Hardening Android

Cahyo Darujati^{1†} and Moh Noor Al Azam^{2††},

cahyo.darujati@narotama.ac.id noor.azam@panigra.id

University of Narotama, Indonesia; PT Panca Anugrah Integrasiindo

Summary

The Sambodana project is a mobile communication security system development project using Hardening Android. The initial idea for this project is that information leakage occurs outside of a communications application with end-to-end cryptographic security. Android hardening prevents unwanted applications and bloatware from being installed, such as unavailable Google Play Store or install restrictions.

Keywords:

Hardening, Android, securing, Sambodana, Naro OS.

1. Introduction

The security of multimedia communications (both voice and data) has become a national concern. One of the government's efforts is to force various popular foreign companies such as WhatsApp[1], Facebook[2], Instagram[3], Twitter[4] and Tiktok[5] to open branches in Indonesia and set up data centres in Indonesia. But one thing that's been forgotten is that conversational evidence and other data are becoming a new source for investigating more information about our behaviour[6]. This has become an important issue in modern times. Considering the existing issues of national affairs and national strategic issues in the field of defence and security, we believe that it is necessary to develop mobile communication security using hardening android[7] with name code Naro OS with logo in Figure 1.



Figure 1 Naro OS Logo

This activity creates a private and secure communication system, as the developed system consists of his two parts: software in the form of applications and operating systems. This application is used to communicate securely while using the Android hardened operating

system, as it uses encryption[8]. The system is expected to be a solution for securing the communication needs of many organizations, but especially partners. This system is the answer to your defence and security needs. The innovation by developing mobile communication security using the hardening android is currently not protected by intellectual property and is still closed. With reference to the downstream system development model, we register intellectual property, starting with trademark registration, design, and patent[9].

The Sambodana project proposing from Narotama University and PT Panca Anugrah Integrasiindo (Panigra). It's about forming a collaboration of both. We can support your organizational and government needs. This collaboration will address the needs of national defences and security issues related to communications security and enable enhanced application of open software development methodologies among the public, especially academics, consultants, practitioners, industry and government. The purpose is that. It is also expected that this activity will create a start-up specializing in communication security that can compete globally in Indonesia.



Figure 2 Sambodana Project Logo

Matching Fund program from kedaireka[10] funding this activity has been combined with hardening android and communication application into a new start-up company with Panigra as a partner. Panigra plans to create a communication system by providing multiple information technology devices including computers and gadgets, but the developed communication system needs many improvements such as chat application security and the operating system used. These developments and improvements have not yet been certified as intellectual property rights but are in line with the downstream research of the Narotama University team. This matching fund activity facilitates collaboration between the applicant and

Panigra develops software development methodologies to become globally competitive start-ups. Development Stages shown in Figure 3.

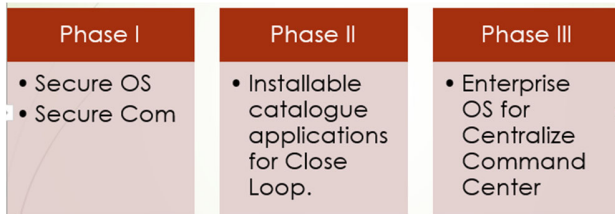


Figure 3 Development Stages

2. Methodologies

One of the differences from typical multimedia telecom companies is the enhanced Android operating system[11] and telecom application integration. The Sambodana solution business model offers the novelty of being able to run multimedia communication services securely and exclusively. Groups or organizations using the Sambodana solution are treated exclusively with respect to multimedia communications to ensure security.

2.1 Android Open Source

Using the list, a portion of manuscript, use the style 'itemize' specified from the style category dropdown menu. One of the main features of Android is that it is open source. The complete operating system source code, including kernel, user interface, libraries, and major applications, is freely available[12]. This means that anyone with the right technical skills can build hardening android from source and flash it onto a compatible device. This flexibility allows different groups, some businessmen and some hobbyists to develop different android distributions. These are often called custom ROMs, but a more appropriate name is custom firmware. You may be wondering how hard it is to create your own custom ROM, a custom version of Android, with all the basic components you need[13]. It is actually possible. Before delving into the dark world of Android's custom builds, we need to pause, keep our expectations in check and appreciate the enormity of the task at hand. If you have no programming experience, no Linux experience, or do not know what a makefile is, this is not for you. Android is a complete operating system. It is complex and contains many different subsystems. Developing an operating system as complex and useful as Android didn't happen overnight. This means that if you do any customization, you should start small. It takes hours of hard work and dedication to create a radically different alternative Android distribution. If you're comfortable writing code and know how to use makefiles and compilers, building your own custom Android ROM can be a rewarding experience.

2.2 AOSP System Requirements

In theory, it's possible to create custom Android firmware for any computing device that can run the latest operating system. However, to make your life easier, we'll stick to building Android for Google's Pixel phones. To build Android, you must have access to Linux and be familiar with it. Since you will be using the terminal a lot, you should be familiar with shell commands. You may be able to use a Linux virtual machine, but we recommend using a dedicated PC. Requires at least 400 GB of storage and 16 GB of RAM or more, but 32 GB or 64 GB is recommended. A modern multi-core 64-bit processor is required. According to Google, building Android on a 6-core computer with 64 GB of RAM takes about five times as long as a 72-core computer with similar RAM. In my tests, he foolishly tried to build a dual-core machine with a 10-year-old CPU. It took about 24 hours! Learn Patience Android development is not fast. Synchronizing the source repository may take several hours depending on the speed of your internet connection. Moreover, a perfectly clean assembly takes several hours. Even after small changes, you may have to wait 10-20 minutes for the build to complete. It all depends on your hardware. Don't expect a new version of Android to be up and running anytime soon. The Android Open Source Project (AOSP)[14] version does not include Google services. So no Google Play, YouTube, Gmail, Chrome, etc.

2.3 Start Creating Your Own Android ROM

The basic flow of custom ROM[15] creation is as follows. Download and build Android from the AOSP. Then modify the source code to get your custom version. However, Google has excellent documentation on building AOSP. Don't skip steps or read through snippets, here are the general steps:

1. Set up your build environment, including installing the appropriate development tools.
2. Get the source. This is done using the "repo" tool.
3. Get Proprietary Binaries - Some drivers are released in binary form only. Select a destination using the "Lunch" tool.
4. Start building with 'm'.
5. Build a bottle on your device using ADB and Fastboot.

Set up your build environment - The recommended build OS is Ubuntu 22.10. You will need to install various development related packages such as gcc, git and python[16]. AOSP's Android master branch includes a pre-built version of his OpenJDK, so no additional installation is required. However, older versions of Android require a separate JDK installation. You should use OpenJDK. To write AOSP, you need to use Python3.

Get your own binary - Binaries are provided as self-extracting scripts. Unzip the archive and run the self-extracting script at the root of the source tree. Binaries are installed in the Vendor/ directory. Note that many Pixel devices have two sets of his binaries from Google and Qualcomm. you need both. Download the version of the binaries that matches the AOSP version you are building[14].

Pick a target - Browse through this list of Pixel[17] devices and select the appropriate build name. For example, if you have a Pixel 5, use the redefinition usability bug.

Start Build - M starts a build. The build system can run multiple jobs simultaneously using the -JN argument. If you don't use the yes argument, the build system automatically chooses the optimal number of tasks for your system.

Flash the build to the device - Flashing the build to a physical device requires Google's platform tools. You can use Google's Android SDK platform tools or find adb and fastboot in ./out/host/linux-x86/bin.

Flashing - After successfully flashing it to the device using "fastboot flash all -w" you will see the vanilla version of his AOSP. There are no Google services or Play Store, just a few core apps. This is the basic structure of Android. But congratulations. You have successfully built Android from source code and flashed it to your device. it's nothing.

Customization Once - Android is up and running, you can start customizing it and creating your own custom ROM. This is where it gets really difficult. You're about to tinker with the innards of the Android operating system, and the problem is, Android is huge. My working directory contains over 350 GB of data including source code, graphics, compiled binaries and tools. It's a lot of things. Let's get started. I will show you two simple tweaks to become an Android firmware hacker.

Customize your Messages app - A fairly simple customization is to modify one of the predefined apps. If you're developing a full replacement Android distribution, it's easy to modify or replace parts of your core app. In this case we are only optimizing, but the principle is the same for more complex changes and revisions. The main apps are in the ./packages/apps/ directory and the messaging apps of interest are in ./packages/apps/Messaging/. Look in res/values/ and edit strings.xml. Edit with your favorite graphics editor. If you want to use the command line, use vi or nano. strings.xml contains all the English texts of the application. If you use another language, you'll need to find the corresponding XML files in the res/ directory. Find the place in string.xml where conversation_list_empty_text is

defined. The hard part starts with "Once you start". Replace with your own string.

Save the file and start another build with the m command. Once the build is complete, flash it to your device and reboot. Launch your messaging app and look for a new text. Of course, this is a simple change, but you have the option to change the default apps to your liking.

Customization Any self-respecting custom Android distribution should contain information about the ROM itself. To do this, you can change the system properties. Locate the sysprop.mk file in the build/core/ directory. Edit it and find the line that defines BUILD_DESC. Save the file, rebuild your device's firmware, and flash it again. After rebooting, go to Settings then About Phone and scroll down.

3. Results

Official Android releases happen about once a year. They are well supported by the Android ecosystem and used by hundreds of millions of mobile his devices. So, you might think there is little reason not to use the official version of Android on your device. Why use a custom mobile OS instead?

3.1 Adaptation means Optimization

Perhaps the most obvious reason to build a custom mobile operating system is that it can be customized to your organization's needs and priorities. However, what is often overlooked is that customization is more than just tweaking the appearance and behaviour of the operating system. Customizations also provide opportunities to optimize operating system performance, such as removing libraries not needed for your use case. You can also customize the OS to improve security by removing unnecessary components (thus reducing the attack surface size) and adding other OS hardening features that are not part of stock Android. I can do it. In short, a custom operating system allows you to create an operating system that outperforms generic Android in terms of UI/UX, performance, and security.

3.2 Deploy OS's on Your Own Schedule

Relying on official Android releases increases pressure to roll out operating system updates to devices based on Android's own release schedule. Of course, there is no law that requires you to update your operating system every time Android releases a new version. However, if the upgrade takes too long, you run the risk of compatibility issues. So basically, you're committing to Android's own schedule for operating system upgrades. However, starting with Android 9, Google is cracking down on OS updates, forcing consumer devices and MDMs to release updates

more frequently, and forcing them to update after 90 days. Additionally, you may need to plan your application deployments and upgrades to align with Android's official release schedule. You probably don't want to publish a new application on the same day as an Android update, as it increases the risk of problems with the new, untested operating system. With a custom operating system, you're no longer tied to the official Android release schedule. Update your operating system and applications on a schedule that best suits your business.

3.3 Save Money

Considering Android is free, you might not think you can save money with a custom mobile operating system. If the official version of Android doesn't cost money, how can you save money by creating your own custom version? It's true that you may not save on direct OS development costs. But in the long run, a custom operating system can save you money by allowing you to create a deployment environment that fits your needs. You can reduce the time and cost of building and deploying applications by reducing the complexity of your environment or by integrating components into the operating system that would otherwise have to be installed separately.

3.4 Platform Independent

Even if you base your custom operating system on Android, you can avoid being locked into the standard Android platform. If you have your own custom codebase, you don't have to worry about how changes in Android itself affect it. For example, let's say Android announces that an upcoming official release will include a software library that won't work on the specific hardware you need or is incompatible with one of your apps. Or, even worse, imagine the Android project changing its licensing policy in questionable ways. Recently, when Elastic changed their licensing policy, we learned that open source license changes could turn the community upside down. If this happens, you're still free to use your existing custom OS code to do whatever you want. You don't have to worry about other people's decisions eroding your ability to deploy the mobile hardware and software your business needs.

3.5 Easier Monitoring and Management

Having full control over the operating system running on your device makes it easier to monitor and manage. Stock Android only allows you to collect monitoring data and use Android-supported management frameworks. Custom operating systems let you extend Android's native management capabilities. You can include additional data in your operating system's device profile to help identify each device in your fleet and track its update history. You can also install custom application deployment tools. This

is safer and easier than what Android or your device provider offers by default.








Name	↑
 Asset-Naro-OS.tar.xz	
 boot-naro-11.1-20221006-EMPU-aliioth.img	
 boot-naro-11.1-20221006-EMPU-aliioth.img.zsync	
 naro-11.1-20221006-EMPU-aliioth.zip	
 naro-11.1-20221006-EMPU-gta4xl.zip	
 naro-12.1-20221021-EMPU-aliioth.zip	
 recovery-naro-11.1-20221006-EMPU-gta4xl.img	

Figure 4 Images ROMs List

3.6 Images for ROMs

In Figure 4, we can see the file naro-12.120221021-EMPU-aliioth.zip which shows that the build of Android ROM version 12 has been successful. 3.7 Flashing Naro OS. Now, we flashing device POCO F3 using boot-naro-11.1-20221006-EMPU-aliioth.img file. Power off the device and boot it into bootloader mode. This step is very crucial, power off device with hold Volume Down Button and Power. Keep holding both buttons until appear the word FASTBOOT on the screen, then release. Flash the image file to device by typing “fastboot flash boot-naro-11.1-20221006-EMPU-aliioth.img”. After task complete, power off and then power on again. If failed, connect the device to PC via USB if it isn't already. When device isn't already in fastboot mode, on PC, open a command prompt (on Windows) or terminal (on linux or macOS) window, and type: “adb reboot bootloader”. Once the device is in fastboot mode, verify your PC finds it by typing: “fastboot devices”

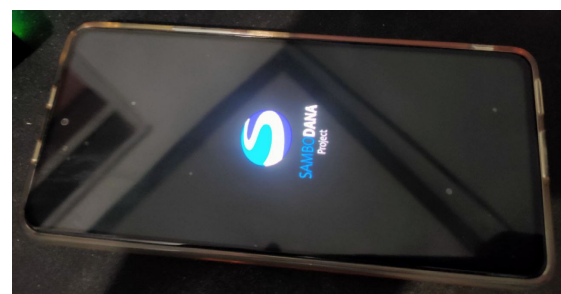


Figure 5 Sambodana Project Splash on POCO F3

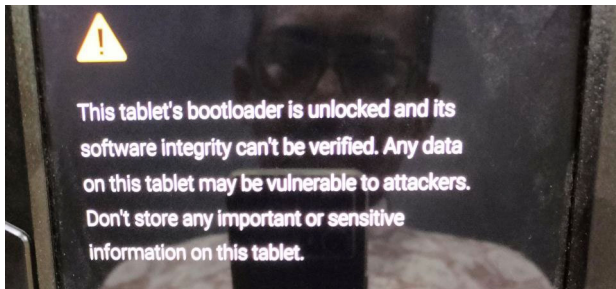


Figure 6 Samsung s6 detect bootloader is unlocked

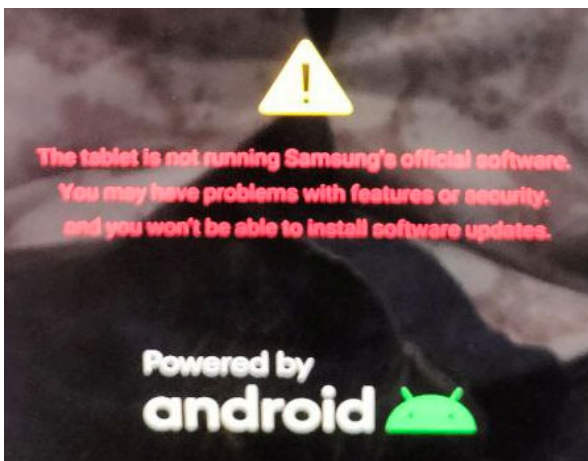
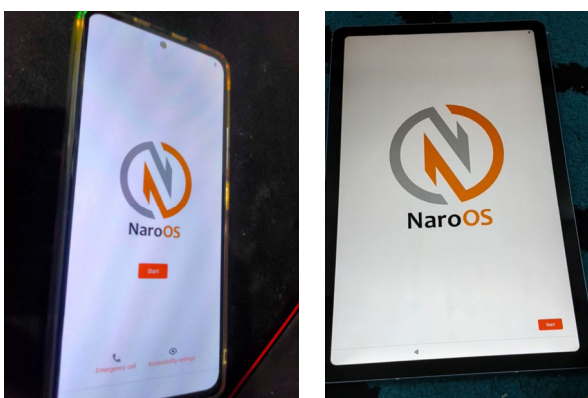


Figure 7 Samsung s6 detect not running samsung's official software

3.7 Booting Naro OS

Appearances whenever first time boot for installation from 2 types devices are different. In Figure 5, OPPO F3 have no problem to display Sambodana Project boot Splash but Samsung Galaxy Tab S6 failed and display messages in Figure 6 dan 7.



(a) Poco F3 (b) Samsung Galaxy S6
Figure 8 Naro OS powered by Hardening Android

3.8 First Installation Naro OS

In figure 8, Both devices very smooth to completed the first Installation Naro OS . Step for the first installation is (i) Language (Bahasa or English), (ii) Date and Time, (iii) Select Wifi Network, (iv) SIM Card Missing announce if detect no SIM Card, (v) Location Services (allow or not), (vi) Update Naro OS Recovery if it necessary, (vii) Naro OS Features, (viii) Protect Your Tablet, (ix) Restore Apps and Data, (x) Finish.

4. Conclusions

Naro OS is a mobile operating system based on Android 12 using hardening techniques such as removing pre-installed applications, Google Playstore. Added an exclusive chat application with end-to-end encryption techniques.

Next, we will build mobile device management and installable catalog applications in private to serve private organizations and private schools.

Acknowledgments

We would like appreciate to the Matching Fund Program 2022 Batch 4 from Kedaireka, Directorate General of Higher Education, Ministry of Education, Culture, Research and Technology of the Republic of Indonesia for this grant and also great collaboration with our industrial partnership, PT Panca Anugrah Integrasindo.

References

- [1] A. Kousar, S. Memon, and I. Ali Simming, "The pragmatic analysis of emojis by ESL learners in Verbal modalities: A case study of Whatsapp chat," *IJCSNS International Journal of Computer Science and Network Security*, vol. 20, no. 10, 2020, doi: 10.22937/IJCSNS.2020.20.10.25.
- [2] D. Alghamdi, "On the Scale in the Kingdom of Saudi Arabia: Facebook vs. Snapchat," *IJCSNS International Journal of Computer Science and Network Security*, vol. 21, no. 12, 2021, doi: 10.22937/IJCSNS.2021.21.12.19.
- [3] L. v. Casaló, C. Flavián, and S. Ibáñez-Sánchez, "Influencers on Instagram: Antecedents and consequences of opinion leadership," *J Bus Res*, vol. 117, 2020, doi: 10.1016/j.jbusres.2018.07.005.
- [4] M. Haffner, "Twitter," in *Geographies of the Internet*, 2020. doi: 10.4324/9780367817534-20.
- [5] C. Montag, H. Yang, and J. D. Elhai, "On the Psychology of TikTok Use: A First Glimpse From Empirical Findings," *Frontiers in Public Health*, vol. 9, 2021. doi: 10.3389/fpubh.2021.641673.

- [6] W. Mazurczyk, L. Caviglione, and S. Wendzel, "Recent Advancements in Digital Forensics, Part 2," *IEEE Security and Privacy*, vol. 17, no. 1, 2019. doi: 10.1109/MSEC.2019.2896857.
- [7] L. A. Aljeraisy and A. Alsultan, "Android Operating System: Security Features, Vulnerabilities, and Protection Mechanisms," *IJCSNS International Journal of Computer Science and Network Security*, vol. 22, no. 11, 2022, doi: 10.22937/IJCSNS.2022.22.11.53.
- [8] N. K. Kamarudin, N. S. Bismi, N. H. Ahmad Zukri, M. F. Mohd Fuzi, and R. Ramle, "Network Security Performance Analysis of Mobile Voice Over Ip Application (mVoIP): Kakao Talk, WhatsApp, Telegram and Facebook Messenger," *Journal of Computing Research and Innovation*, vol. 5, no. 2, 2020, doi: 10.24191/jcrinn.v5i2.136.
- [9] L. Laksmindari, "Kebijakan Pemerintah Dalam Perlindungan Hak Kekayaan Intelektual (HAKI) Di Indonesia," *Pencerah Publik*, vol. 5, no. 2, 2018, doi: 10.33084/pencerah.v5i2.1012.
- [10] N. Rahmawati and A. Suzianti, "Development of Customized Balanced Readiness Level Assessment Prototype for Research Funding Instruments".
- [11] P. Gilski and J. Stefanski, "Android OS: A Review," *TEM Journal*, vol. 4, no. 1, 2015.
- [12] Y. Aafer, X. Zhang, and W. Du, "Harvesting inconsistent security configurations in custom android ROMs via differential analysis," in *Proceedings of the 25th USENIX Security Symposium*, 2016.
- [13] M. Suleman, X. Zhong, and Y. Sun, "Empirical Research and Auxiliary Tool for Custom Android ROMs," in *Proceedings - 2020 International Symposium on Computer Engineering and Intelligent Communications, ISCEIC 2020*, 2020. doi: 10.1109/ISCEIC51027.2020.00011.
- [14] S. S, "Developing Custom ROM based on Android using AOSP," *Int J Res Appl Sci Eng Technol*, vol. 8, no. 8, 2020, doi: 10.22214/ijraset.2020.30932.
- [15] T. B. MohdAnis and S. Subramaniam, "Operating System: The Power of Android," *International Journal of Science and Research (IJSR)*, vol. 3, no. 11, 2014.
- [16] D. Yugandhar and J.Kiran Kumar, "A Study on Current Mobile Operating Systems," *Int J Sci Eng Res*, vol. 8, no. 5, 2017.
- [17] W. Song *et al.*, "Towards Transparent and Stealthy Android OS Sandboxing via Customizable Container-Based Virtualization," in *Proceedings of the ACM Conference on Computer and Communications Security*, 2021. doi: 10.1145/3460120.3484544.



Cahyo Darujati received a doctor in 2020 from Electrical Engineering Department, Institut Teknologi Sepuluh Nopember, Surabaya, Indonesia. He is an Assistant Professor Faculty of Computer Science, Universitas Narotama, Surabaya, Indonesia. His research interest includes computer vision, image processing, and security. He is IAENG member, leader OWASP Surabaya Chapter and Cybersecurity Indonesia.