Creating a Secure VoIP Stack for Android Using SELinux

Fotios Lindiakos
flindiak@gmu.edu

Burns Mijanovich
bmijanov@gmu.edu

I. Introduction

The objective of this project was to enhance the security of an Android Voice over IP (VoIP) stack by utilizing Security Enhanced Linux (SELinux).

II. SELinux

SELinux is a Linux feature that provides a mechanism for supporting access control security policies, including mandatory access controls, through the use of Linux Security Modules (LSM) in the Linux kernel. It is not a Linux distribution, but rather a set of Kernel modifications and user-space tools that can be added to various Linux distributions. Its architecture strives to separate enforcement of security decisions from the security policy itself and streamlines the volume of software charged with security policy enforcement

SELinux can potentially control which activities are allowed for each user, process and daemon, with very precise specifications. However, it is mostly used to confine daemons like database engines or web servers that have more clearly defined data access and activity rights. A confined daemon that becomes compromised is thus limited in the harm it can do. Ordinary user processes often run in the unconfined domain, not restricted by SELinux but still restricted by the classic Linux access rights. [1]

III. SE Android

Security Enhanced Android (SE Android) is a project to identify and address critical gaps in the security of Android. Initially, the SE Android project is enabling the use of SELinux in Android in order to limit the damage that can be done by flawed or malicious apps and in order to enforce separation guarantees between apps. The current SE Android reference implementation provides a worked example of how to enable and apply SELinux at the lower layers of the Android software stack and provides a working demonstration of the value provided by SELinux in confining various root exploits and application vulnerabilities. [2]

IV. Approach

Our approach was to first implement a secure VoIP stack on the phone by creating a virtual private network (VPN) connection to a simulated enterprise. We accomplished this by creating a virtual maching on Amazon's EC2 service which was running OpenVPN[1] and configured to only allow VPN network traffic. This virtual machine had one interface on the Internet and another on a Local Area Network (LAN) segment with no other connections to the Internet. This LAN segment was where we placed users who were connected via VPN. We then created a virtual machine running FreePBX[2] to act as our Session Initiation Protocol (SIP) server, and attached it to the LAN segment as well.

After the servers were set up, we downloaded the Android source code from the Android Open Source Project (AOSP) for the android-4.0.3_r1.1 branch. We compiled our own kernel as well as created a system image with the standard Ice Cream Sandwich Operating System as well as the standard applications. We then installed the OpenVPN client application and added our relevant keys. Next we installed and configured Sipdroid[3] to connect to our SIP server over the VPN tunnel.

After that was configured and operating correctly, we obtained the SE Android code for the same

[1]http://openvpn.net/
[2]http://www.freepbx.org/
[3]http://code.google.com/p/sipdroid/

Android branch. From this, we generated a new kernel and userspace, which included the SELinux enhancements.

We then reinstalled the OpenVPN and Sipdroid applications and booted the emulator with our new kernel and system image. After we verified that the applications ran correctly, we modified the `sepolicy` to label our VPN interface, `/dev/tun` with a type of `tun_device` in the `file_contexts` file. We also added the OpenVPN application to the `seapp_contexts` file so that we could create a new domain for the application. We then regenerated our sytem image so that the new rules ensured that the device and application were properly labelled.

After booting the phone, we were then able to run our applications and obtain the Access Vector Cache (AVC) denial messages from the OpenVPN client attempting to manage the tunnel device. We used those messages and the `audit2allow` tool to generate an even more extensive policy which contained all of the actions OpenVPN was trying to perform, and added that to the `app.te` file.

After regenerating the system image again, we no longer got any AVC denial messages when running the applications. We must note here that we did not have to modufy any characteristics of the Sipdroid application because it's requests were made through `binder` which strips any SELinux contexts from requests. This effectively prevents any network based labelling and permissions.

V. TOOLS

In order to better facilitate building our kernel and system images, as well as running our emulators, we developed a set of scripts to simplify the process. We also gathered the patches we needed in order to get the build compiling successfully and included them in our repository so we can more easily duplicate our efforts. We published all of the source on Github and hope to improve them and contribute them to the Android community.[3]

VI. CONCLUSION

In conclusion, we learned a significant about Android and SELinux throughout this process. While we were disappointed that the network level labelling that exists in Linux is not yet implemented in the Android version of SELinux, we are hopeful that when it is, our project will be a reality.

We think that this is a first step in the direction of allowing organizations to isolate their crucial business applications on untrusted devices. This is important as many organizations want to use commodity hardware and open source software for their trusted devices.

REFERENCES

[1] Wikipedia, "Security-enhanced linux — wikipedia, the free encyclopedia," 2012, [Online; accessed 14-May-2012]. [Online]. Available: \url{http://en.wikipedia.org/w/index.php?title=Security-Enhanced_Linux&oldid=491028541}

[2] S. Wiki, "Seandroid — selinux wiki,," 2012, [Online; accessed 13-May-2012]. [Online]. Available: \url{http://selinuxproject.org/w/?title=SEAndroid&oldid=1170}

[3] F. Lindiakos, "GitHub - fotioslindiakos/seandriod_tools," https://github.com/fotioslindiakos/seandriod_tools, 2012. [Online]. Available: https://github.com/fotioslindiakos/seandriod_tools