Análisis y caracterización de frameworks para detección de aplicaciones maliciosas en android

Conference Paper · June 2014

CITATIONS
3

READS
753

5 authors, including:

Andres Navarro
University ICESI
101 PUBLICATIONS **183** CITATIONS
SEE PROFILE

Sebastian Londoño
University ICESI
9 PUBLICATIONS **26** CITATIONS
SEE PROFILE

Christian Camilo Urcuqui López
University ICESI
17 PUBLICATIONS **12** CITATIONS
SEE PROFILE

Análisis y caracterización de frameworks para detección de aplicaciones maliciosas en android

Navarro, Andres., Londoño, Sebastián y Urcuqui, Christian.
{anavarro, slondono, ccurcuqui } @icesi.edu.co
Grupo de Investigación i2t – Universidad ICESI

Fuentes, Manuel y Gomez, Johan
{manuel.fuentes, johan.gomez} @password.com.co
Password – Seguridad Informática

Resumen—Android incorpora distintos sistemas de seguridad para solucionar los problemas generados por aplicaciones malware como el modelo de permisos para cada aplicación, su propia plataforma de distribución de aplicaciones móviles "Google Play", entre otras. Pero existen otras alternativas que tienen como función mejorar la protección contra el software malicioso. En este artículo se presenta la información más reciente sobre metodologías, frameworks y sistemas que permiten resolver algunos de los problemas generados por aplicaciones maliciosas (Malware) para dispositivos móviles con sistema operativo Android. Por medio del estudio descriptivo anterior, se presentan los temas aún por abordar a nivel de la seguridad y finalmente se describe una nueva plataforma de análisis, validación y configuración de aplicaciones Android que implementa el análisis estático, dinámico y una mejora del ASEF el cual tiene como nombre SafeCandy.

Índice de Términos— Android, maliciosas, detección, SafeCandy, estático, dinámico.

I. INTRODUCCIÓN

Android es un sistema operativo de código abierto basado en el kernel de Linux para dispositivos móviles, con un buen posicionamiento en el mercado pero con graves problemas de seguridad para sus usuarios. Android implementa un modelo de seguridad que se subdivide en dos capas, la primera que hace referencia a un modelo de permisos a nivel de la aplicación y la segunda a un aislamiento de los accesos al kernel. Adicionalmente Google cuenta con su propia plataforma de distribución digital de aplicaciones móviles "Google Play", por la cual se implementan otros procesos de seguridad sobre las aplicaciones con el fin de limitar la difusión de código malicioso (Malware). Las publicaciones sobre amenazas móviles no son del todo recientes, pero la mayoría de los Malware tienen como objetivo obtener la información de los usuarios ya sea datos de sus contactos, redes sociales, interceptar llamadas, cuentas bancarias, entre otras. Entre algunos de los casos representativos de ataques de Malware para Android se encuentran, Dendroid Toolkit el cual permite generar archivos maliciosos APK personalizables, Android.MisoSMS implementa una encriptación XTEA en los mensajes capturados con el fin de reducir su nivel de detección por los sistemas de seguridad, entre otros casos. Para dar solución a los problemas generados por los Malware, se han propuesto técnicas para evaluación de las aplicaciones entre las cuales se encuentran, el análisis estático, análisis dinámico y el aprendizaje automático, por otra parte también se han desarrollado frameworks para mejorar la seguridad en dispositivos móviles Android, como el Android Security Evaluation Framework (ASEF), Static Android Analysis Framework (SAAF), WallDroid, etc.

Entre algunos sistemas de seguridad para aplicaciones Android se encuentra Mobsafe, el cual permite realizar el análisis basado en una plataforma de computación en la nube y minería de datos en un tiempo significativo. Pero como anteriormente se ha descrito aún no existe algún sistema capaz de solucionar todos los problemas presentados por los Malware; para abordar la problemática detectada se propone la descripción de una nueva plataforma de análisis, validación y configuración de aplicaciones Android que implementa el análisis estático, dinámico y una mejora del ASEF el cual tiene como nombre

SafeCandy.

II. Análisis y caracterización

A. *Métodos de análisis*

Una de las formas de enfrentar la creciente propagación de malware en Android es utilizando métodos estáticos y métodos dinámicos para la detección de la aplicaciones maliciosas.

B. *Métodos de análisis estático*

Este método consiste en la utilización de herramientas para obtener el código fuente de la aplicación y los archivos o datos que componen el apk para posteriormente evaluar las funciones que realiza el malware y qué nivel de peligrosidad poseen [7].

A través de una novedosa metodología llamada SOM (Self Organizing Map) [8], se provee una visualización en 2d que identifica la correlación entre los permisos, esto permite correlacionar llamados y funciones de las aplicaciones en Android para determinar comportamientos irregulares. Por otro lado se encuentran herramientas como TaintDroid [9] que notifica al usuario acerca de aplicaciones que desean acceder a permisos que están en la lista negra. Mediante la utilización de un descompilador Dalvik analizan el código fuente en java a través de su archivo de instalación.

Stowaway[10] es una herramienta que detecta sobre-privilegios (overprivilege) en las aplicaciones de android, con ésta se evaluaron distintas aplicaciones de distintos marketplaces y se concluyó que más de un tercio de las aplicaciones publicadas están con sobre-privilegios, los resultados revelan que la mayoría de desarrolladores están tratando de seguir los principios de privilegios bajos pero fallan por falta de información confiable acerca de los permisos.

Otras implementaciones consisten en herramientas de análisis, para construir grafos de dependencia estática con los llamados a procedimientos internos de conectividad. Los desarrollos capturan la relación de consumos de información de las aplicaciones, a través de la identificación de los caminos de la información obtenida por las entradas (inputs) y los métodos de puntos de acceso para servicios críticos del sistema.

Crowdroid[11] es una propuesta que busca los llamados open(), read(), access(), chmod() y chown(), que son los llamados más utilizados por aplicaciones malware. SAAF[12] provee un análisis del flujo de datos en la aplicación y un gráfico del flujo de control.

Airmid[13] utiliza la colaboración entre los sensores in-network y los dispositivos inteligentes para identificar la procedencia de tráfico malicioso. La herramienta cuenta con opción de reparación remota que valida con un sistema de detección localizado en un servidor externo el software malicioso.

C. *Métodos de análisis dinámico*

Estos métodos estudian el comportamiento del malware en ejecución mediante la simulación de gestos, se analiza la interfaz de usuario, procesos de ejecución, conexiones de red, apertura de sockets. El objetivo de este análisis es generar gestos que permitan detectar un comportamiento malicioso [14].

Paranoid Android es un sistema donde investigadores pueden ejecutar un análisis completo de aplicaciones maliciosas en la nube utilizando réplicas de teléfonos móviles [15]. DroidMOSS[16] utiliza técnicas de lógica difusa para eficientemente localizar y detectar cambios en el reempaquetado de las aplicaciones.

D. *Machine learning*

Schmidt[17] propone una solución basada en el monitoreo de eventos en el kernel de linux, Mediante la utilización de la herramienta “readelf” se lee la información contenida en los ejecutables y se clasifica. Luego se aplica la herramienta para analizar las funciones llamadas en las aplicaciones normales y en las maliciosas, a través de 3

clasificadores PART (extracción de las reglas de decisión del árbol de clasificación), Prism (una regla simple de inducción que cubre todo el set de reglas) y nNb (una versión liviana del algoritmo del vecino más cercano), se predice si la aplicación es maliciosa o no.

Frank[18] investiga la diferencia entre las aplicaciones de "alta reputación" y "baja reputación" para la identificación de malware, el método utilizado usa las peticiones de permisos y no utiliza análisis estático a menos que haya código disponible.

Shabtai[19] utiliza un clasificador similar para juegos y herramientas en android como un proxy para la detección de malware.

Sanz[20] aplica distintos tipos de clasificadores a los permisos, ratings y a string estáticos de 820 aplicaciones para ver si podían predecir la categoría de las aplicaciones, utilizando este escenario para la detección de malware por categoría.

Zhou[21] encontró malware a través de DroidRanger, un sistema de detección de malware que utiliza permisos como una entrada.

III. Framework Mobsafe

A través Saturn-cloud[22][23] una plataforma de computación en la nube casera, se realiza la tarea de análisis de seguridad. Saturn-storage, que es un almacenamiento NFS con un sistema de archivos ZFS (openindiana+napp-it) [24][25] es utilizado para acomodar las máquinas virtuales. Puede llegar a utilizar 16 discos duros cada uno con 2TB de almacenamiento tipo SATA, para un total de 32TB. Cloudstack es utilizado para manejar Vmware vSphere. la figura 1 muestra la infraestructura:

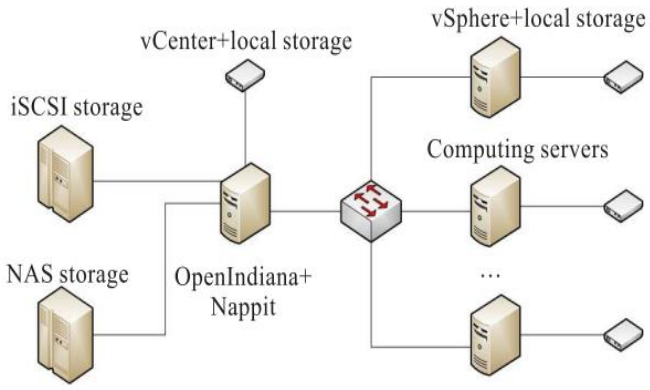

Figura 1

El almacenamiento para las aplicaciones móviles utiliza el sistema de archivos HDFS, cuenta con 40 servidores y 40TB de espacio.

A. Principio de trabajo

Mobsafe es un sistema para chequear si una aplicación de android es de virulencia o maligna basado en unas herramientas en la nube. el procedimiento para análisis de archivos se muestra en la figura 2:

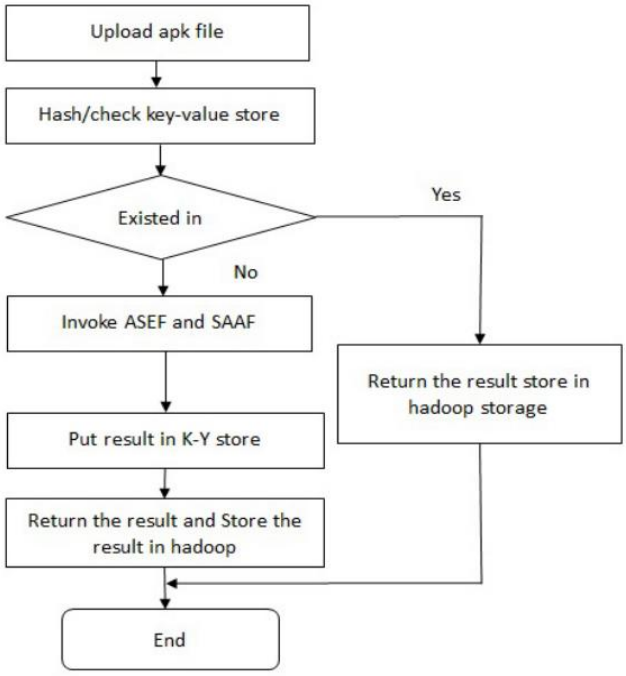

Figura 2

Cuando se presenta un apk para el análisis a través de mobsafe, el sistema chequea el valor de la llave para comprobar si se ha analizado

anteriormente, en caso de que ya haya sido analizado se devuelve el valor del análisis y se termina, en caso de no haberse analizado antes, se procede a invocar los análisis estáticos y dinámicos (ASEF y SAAF) luego se guarda el resultado en el almacenamiento hadoop y termina.

B. Frontend

Para la presentación del Mobsafe, se utiliza SpringSource Spring framework[26] y Twitter Boostrap[27]. Este provee la función de subir archivos y mostrar el resultado del análisis en la web.

C. Backend

ASEF es una herramienta automatizada que permite el análisis de aplicaciones en Android. Cuando se presenta una aplicación desconocida para el análisis ASEF, primero empieza el logging ADB y el sniffing de tráfico a través de TCPDUMP, luego se lanza el AVD (Android Virtual Machine) en el que se instala la aplicación. Se procede a inicializar el ASEF y envía gestos aleatorios para simular la interacción humana en la aplicación. Mientras tanto, ASEF también compara los logs de la máquina virtual de android con la librería CVE y su actividad con Google Safe browsing API[28]. Luego de que han sido enviados un número de gestos a la máquina virtual, el test se finaliza, se desinstala la aplicación y se cierra la máquina virtual. ASEF analiza entonces los logs generados y el tráfico que la aplicación generó. ASEF utiliza Google Safe Browsing API para encontrar si las URLs a las que trató de acceder la aplicación fueron maliciosas o no. ASEF además chequea la vulnerabilidad con una lista de vulnerabilidades conocidas.

D. SAAF

SAAF es un análisis estático para apk de Android. Extrae el contenido del apk y decodifica el contenido a código smali[29], luego aplicará fracciones del código obtenido para analizar los permisos de las aplicaciones, correlacionar patrones heurísticos y realizar cortes de código de funciones de interés.

E. Otras herramientas

Se utilizaron además otras herramientas para análisis estático como readelf[30], ded[31], apktool[32], androguard[33] y soot[34]. La mayoría de estas herramientas utilizan ingeniería inversa. Otras de análisis dinámico fueron Strace[35] y Randoop[36] para detección de aplicaciones basados en el comportamiento en tiempo de ejecución. Strace observa las llamadas en kernel de linux mientras que Randoop estimula la aplicación en android a través de gestos de entrada aleatorios y observa los mensajes de salida.

F. Evaluación

Para la evaluación del sistema se recolectaron aplicaciones desde el 1 de mayo al 31 de julio en el 2012. El tamaño de los datos fue de 1 TB de logs comprimidos zip (el tamaño expandido es superior a los 10TB). Los dispositivos android se clasificaron en 3 categorías: clase baja, clase media y clase alta dependiendo de la resolución de la pantalla.

G. Pruebas de rendimiento

1. ASEF

Se utilizaron 20 aplicaciones descargadas de AppChina, analizar cada aplicación demoró entre 64s a 150s, y el tiempo promedio fue de 100s. El análisis completo se demora menos de 2 minutos en promedio. La mayor parte del tiempo (80%) es consumida en la instalación y testeo de la aplicación como se muestra en la figura 3. Si se requiere reducir el tiempo total, se debería tratar de acelerar estos dos pasos. Entre más gestos tenga el test, más se demora el análisis completo. Cuando se disminuyeron los gestos de 1000 a 200, el tiempo total disminuyó 20s.

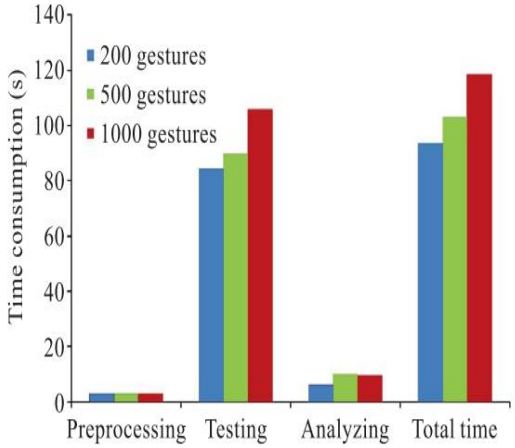

Figura 3

2. SAAF

Se analizaron 25 aplicaciones del marketplace AppChina, la mayor parte del tiempo en el SAAF se consume en el corte del código y también en la categorización de los permisos. El tiempo promedio para analizar una aplicación con una máquina Linux, con procesador Intel-i5 con 4GB de RAM es de 33.93s

H. Estancias estimadas

Si se realizara el test ASEF para 800.000 aplicaciones en total se consumirían 450 horas con 50 máquinas virtuales, cada una de ellas con procesador core-i5 y 4Gb de memoria ram. Si se aplicara el test SAAF para la misma cantidad de aplicaciones se consumirían 151 horas con 50 máquinas virtuales con las características previas.

El promedio de aplicaciones instaladas en un dispositivo Android es de 30, cuesta alrededor de 1 hora utilizar ASEF y SAAF para finalizar el análisis en una máquina virtual y el AVD. Pero si se distribuye la instalación de las aplicaciones en máquinas virtuales o AVDs separadas, el tiempo total puede ser menos de 1 minuto, lo que es aceptable para la experiencia de usuario en cuanto a chequeo de seguridad.

IV. FRAMEWORK WALLDROID

Las pruebas de seguridad son una actividad difícil de desarrollar, esto es porque a diferencia de las pruebas de funcionalidades específicas de la aplicación que buscan cumplir requerimientos y especificaciones, las pruebas de seguridad son una forma de "pruebas negativas" (negative testing) que muestran un cierto comportamiento que no existe en el sistema.

Una forma automatizada de pruebas de seguridad que no requiere casos de prueba o un esfuerzo significativamente importante es llamado "prueba de exploración de vulnerabilidad" (fuzz testing) [37]. En resumen, fuzzing es una forma de pruebas negativas que se alimenta a través de entradas malformadas e inesperadas con el objetivo de revelar vulnerabilidades en la seguridad.

En el pasado, el "fuzzing" ha sido empleado por la comunidad hacker como uno de los métodos predominantes para romper un sistema y ha mostrado resultados exitosos.

En el caso de android, a través del método "fuzzing" se encontró una vulnerabilidad activada simplemente por la recepción de un tipo particular de mensaje SMS, que no solamente terminaba el proceso del teléfono sino que también sacaba el objetivo de la red[38].

Actualmente se pueden encontrar fuzzers disponibles, uno de ellos es Android Monkey[39][40], el cual genera casos aleatorios de pruebas, generalmente no son efectivos en la práctica. Por otro lado, las pruebas "fuzzing" son generalmente consideradas costosas en términos de consumo de máquina, esto también depende de la cantidad de gestos que se deban generar.

Se desarrolló una versión escalable para un fuzzing inteligente para las pruebas de aplicaciones Android. El framework es escalable en términos de tamaño de código y número de aplicaciones mediante el apalancamiento del poder de la computación en la nube. El framework utiliza distintas técnicas de análisis heurísticos y de software para inteligentemente guiar la generación de casos de prueba que potencien el descubrimiento

de vulnerabilidades.

El framework busca responder la siguiente pregunta: dadas las técnicas de pruebas de software y un amplio poder de procesamiento, ¿qué vulnerabilidades de seguridad se podrían descubrir automáticamente?

A. Arquitectura

Como se muestra en la figura 4, se explicará la arquitectura de la solución.

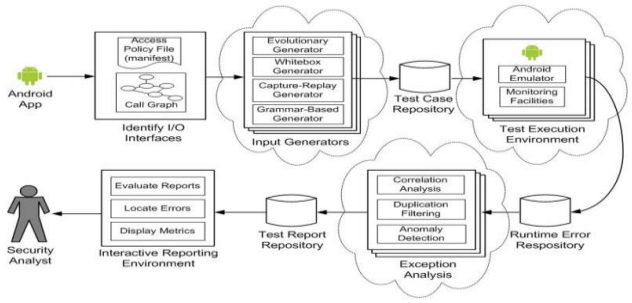

Figura 4

Dada una aplicación para las pruebas, el primer paso es identificar automáticamente las interfaces de entradas y de salida. Las interfaces de entrada de una aplicación representan los distintos caminos en que pueden ser invocadas o por el entorno de ejecución o por el usuario. Estas pueden incluir GUIs, interfaces de red, archivos, APIs, etc. Las interfaces de salida de la aplicación también son importantes, dado que un comportamiento anormal puede llevar a detectar vulnerabilidades.

A través de una variedad de técnicas de análisis, se identifican hasta las interfaces que están ocultas, por ejemplo, se desarrolló una técnica de análisis que permite identificar todos los widgets de la interfaz de usuario a través de los cuales se interactúa con el sistema.

Otra técnica incluye el grafo de llamadas de la aplicación, el árbol abstracto de sintaxis, y el manifiesto que provee mucha meta información acerca de la arquitectura de la aplicación y sus permisos de acceso. Hasta este momento el código fuente del APK no está disponible, se utiliza ingeniería inversa para obtenerla a través de herramientas como dextojar[41].

Luego se procede al motor generador de entradas que crea los candidatos a pruebas, cada uno se apalanca en distintos lineamientos heurísticos para la generación de los casos. Esto permite tener una variedad de casos a probar con distintas características. Dado que algunos generadores son computacionalmente más costosos y toman más tiempo en ejecutarse, el framework ejecuta varias instancias de la aplicación en la nube de forma paralela. Por ejemplo, se revisó el java pathfinder[42] que es un motor simbólico de ejecución desarrollado por la NASA Ames, que permite el desarrollo de casos de prueba para aplicaciones android. La utilización de dicha herramienta permite la ejecución sistemática de aplicaciones en android para la generación de casos de prueba que ejerciten distintas partes de la aplicación, además de proveer una cobertura de código favorable.

Luego de la generación de los escenarios de prueba, se procede al ambiente de ejecución de las mismas que prueba en muchas instancias de la aplicación los escenarios. Se utilizan emuladores de android en la nube. Además se utilizan instalaciones de vigilancia de android (Intent Sniffer[43]) para la recolección de datos en la ejecución. Estos guardan los problemas y errores como excepciones, violaciones de acceso, basura, que surgen en el Runtime Error Repository. Posteriormente, el motor de Exception Analysis investiga los datos obtenidos en el Runtime Error Repository para correlacionar los casos ejecutados con los problemas ya reportados y también potenciales vulnerabilidades de seguridad. Por otra parte, el motor Exception Analysis reduce la información recolectada para filtrar cualquier redundancia, dado que la misma vulnerabilidad puede ser encontrada en más de uno de los casos de pruebas ejecutados.

También se buscan comportamientos anormales como degradación de rendimiento, que puede indicar vulnerabilidades (por ejemplo, una entrada de prueba que puede ser usada para instigar un ataque de denegación de servicios). El resultado de estos análisis es guardado en el Test Report Repository, que luego es usado por el Interactive Reporting Environment para permitir al analista de seguridad evaluar la robustez de la aplicación y entender su vulnerabilidad.

V. FRAMEWORK SAFECANDY

SafeCandy[44] es un sistema con arquitectura cliente-servidor que permite realizar el análisis, validación y configuración de seguridad para aplicaciones Android. El servidor brinda los servicios de protección contra malware a todo tipo de aplicaciones Android de forma remota a través de un aplicativo Android instalado en cada cliente como se muestra en la figura 5, la cual permite la interacción con el usuario y además provee un análisis del dispositivo de manera local. El sistema presenta las siguientes características:

- Protección contra aplicativos malware que puedan acceder a información privada, espías de localización, suscripción a servicios premium a través de SMS sin autorización, entre otros.
- Test realizados por medio de servicios en la nube y de forma local utilizando una base de datos en el dispositivo.
- Detección de vulnerabilidades de la aplicación escaneada.
- Elaboración de reportes a partir de la información recolectada.
- Prueba de aplicaciones en AVD's (Android Virtual Devices).
- Ejecución de los escaneos a partir de eventos programados que tienen como fin mejorar la experiencia del usuario sin alterar el rendimiento del dispositivo, por ejemplo cuando el dispositivo se encuentra recargando su batería, cuando hay conexión WIFI, entre otros.
- Utilización de bases de datos externas sobre aplicativos malware para así realimentar la base de datos del servidor.
- Análisis y validación de una aplicación antes de ser instalada, sin importar que esta provenga de desarrolladores de Google Play u otras tiendas de aplicaciones para Android.

Arquitectura

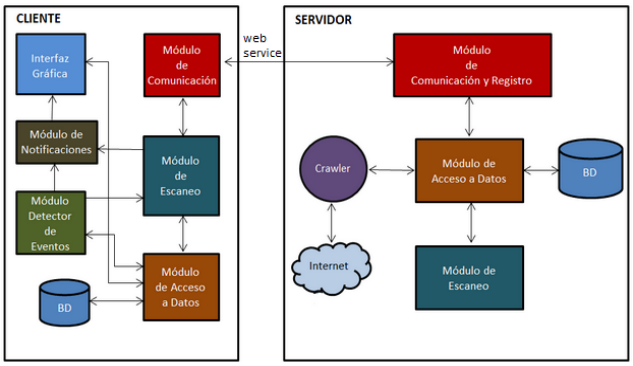

Figura 5

A. Cliente:

Los módulos del cliente contenidos en la aplicación Android SafeCandy son:

- Módulo de comunicación: Este módulo permite la comunicación entre el cliente y el servidor
- Módulo de detección de eventos: Encargado de ejecutar los escaneos necesarios una vez se haya detectado un evento programado.
- Módulo de escaneo: Este módulo es el encargado de realizar los escaneos locales, iniciar los escaneos remotos y actualizar la base de datos del dispositivo.
- Módulo de notificaciones: Encargado de gestionar las notificaciones para el usuario.
- Interfaz gráfica: Es el módulo encargado de controlar la interacción entre el usuario y el aplicativo.
- Módulo de acceso a datos: Permite la persistencia de la información y la comunicación con la base de datos.

B. Servidor:

El servidor de la plataforma cuenta con cinco elementos principales, los cuales son:

- Módulo de comunicación y registro: Este módulo permite recibir y responder las peticiones del cliente a través de un protocolo de comunicación propio.
- Módulo de acceso a datos: Encargado de la comunicación con la base de datos y así mismo la administración de la información.
- Módulo de escaneo: Este módulo es el encargado de gestionar los escaneos (dinámico y estático) de las aplicaciones registradas por analizar y validar.
- Crawler: Es una aplicación que permite la descarga de las aplicaciones desde el Google Play y llevar su registro en la cola de escaneos pendientes.
- Base de datos: contiene la persistencia de la información suministrada por los módulos de comunicación, escaneo y el Crawler. Los datos almacenados representan a las aplicaciones escaneadas o por escanear, los reportes de seguridad, información detallada sobre cada análisis de cada aplicación, lista de permisos, lista de emuladores por test dinámico, entre otra información.

C. Herramientas utilizadas

- Androguard, es una herramienta desarrollada que tiene como función realizar el análisis del malware a partir de ingenieria inversa. Entre sus características se encuentran:
    - Indica los riesgos de una aplicación.
    - Valida si una aplicación se encuentra en una base de datos de malware.
    - Accede al análisis estático del código, como a los permisos, instrucciones, entre otros.
- Apktool, es una herramienta que permite descompilar los archivos contenidos en un apk y así mismo volverlos a compilar.
- filescan.py, es un código que permite hacer un análisis de todos los archivos contenidos en un apk, con la capacidad de identificar archivos comprimidos, archivos DEX, archivos ELF, ejecutables y archivos de texto que contienen direcciones URL, números de teléfono y comandos shell.
- Scanapp.py, es un desarrollo que realiza la mayor parte de los escaneos estáticos haciendo uso de Androguard, Apktool y filescan.py.
- Dex2Jar, permite descompilar los archivos DEX a código java.
- Drozer, asume el rol de una aplicación Android e interactúa con el Dalvik VM con el fin de buscar vulnerabilidades en aplicaciones y dispositivos.
- Servicio Google safe browsing, es un API de Google que permite buscar la reputación y confiabilidad de una URL en su base de datos.
- Servicio WOT, comprueba la reputación y confiabilidad de una URL.
- Servicio PhishTank, permite descargar de base de datos información de URLs phishing
- Android asset packing tool, es una herramienta que permite obtener información del archivo manifiest de cada aplicación.
- Tcpdump, permite la captura de paquetes enviados y recibidos de un ordenador conectado.
- Android Debug Safe, es una herramienta de línea de comandos que facilita la comunicación ya sea a una instancia de un emulador o de un dispositivo con sistema operativo Android.

D. Ventajas

Safe Candy es un sistema que implementa una arquitectura que está en la capacidad de incorporar distintas herramientas y metodologías para la detección de software malicioso a diferencia de otros entornos existentes. Con el fin de descubrir comportamientos inusuales de cada aplicación, nuestra propuesta implementa tanto el análisis dinámico como el estático sobre cajas de arena en AVD´s. Otra de las ventajas es la modificación del

análisis a través del ASEF, con la cual se busca reducir el tiempo de estudio para cada aplicación sin afectar la eficiencia del proceso. A diferencia de otras soluciones desarrolladas en el mercado Safe Candy incorpora un novedoso concepto que tiene como objetivo brindar una buena experiencia a distintos tipos de usuarios (las personas paranoicas y las frescas). Finalmente, por medio de su arquitectura se tiene la capacidad de implementar soluciones en la nube como machine learning y minería de datos.

VI. CONCLUSIONES

En este paper se analizaron distintas herramientas y frameworks para la automatización en la detección de aplicaciones maliciosas en Android, la mayoría de soluciones han optado por un modelo de computación en la nube que permite simular la aplicación en tiempo real y observar su comportamiento. Dicho modelo puede mejorar exponencialmente dependiendo de la capacidad computacional.

Tanto las soluciones de código abierto como las cerradas de virtualización ofrecen prestaciones suficientes para la simulación de los AVD (Android Virtual Device) de manera correcta. La capacidad de procesamiento es un factor fundamental para mejorar los tiempos de análisis.

Los análisis dinámicos y estáticos ofrecen soluciones para la detección de aplicaciones maliciosas aunque como se examinó en capítulos anteriores, uno de los caminos más prometedores para el futuro de las soluciones en la nube con dichas herramientas, es la implementación de métodos complejos de machine learning que permitan efectuar modelos de clasificación dependiendo de los nuevos comportamientos que vayan surgiendo, además de la utilización de minería de datos (data mining) para la toma de decisiones. Otros métodos que se deben utilizar son "componente primario de análisis" (PCA – Primary Component Analysis) y factorización matricial.

REFERENCIAS

[1] (2012). Android, the world's most popular mobile platform | Android. Recuperado de http://developer.android.com/about/.

[2] Hsiao, K. (2013). Android Smartphone adoption and intention to pay for mobile Internet: perspectives from software, hardware, design, and value. *Library Hi Tech*, *31*(2), 216-235.

[3] Smalley, S., & Craig, R. (2013). Security enhanced (se) android: Bringing flexible mac to android. *20th Annual Network and Distributed System Security Symposium (NDSS'13)*.

[4] (2012). Se intensifican los ataques a dispositivos Android - MIT Technology. Recuperado de http://www.technologyreview.es/read_article.aspx?id=39717.

[5] Swati Khandelwal (2014). Android Malware 'Dendroid' targeting Indian Users. Recuperado de The Hacker News. - http://thehackernews.com/2014/03/android-malware-dendroid-targeting_26.html.

[6] Android.MisoSMS : Its back! Now with XTEA | FireEye Blog. Recuperado de http://www.fireeye.com/blog/technical/malware-research/2014/03/android-misosms-its-back-now-with-xtea.html.

[7] Batyuk, L., Herpich, M., Camtepe, S. A., Raddatz, K., Schmidt, A., & Albayrak, S. (2011). Using static analysis for automatic assessment and mitigation of unwanted and malicious activities within Android applications. *Malicious and Unwanted Software (MALWARE), 2011 6th International Conference on*. IEEE.

[8] Kohonen, T. (2001). *Self-organizing maps* (Vol. 30). Springer.

[9] Enck, W., Gilbert, P., Chun, B., Cox, L. P., Jung, J., McDaniel, P., et al. (2010). TaintDroid: An Information-Flow Tracking System for Realtime Privacy Monitoring on Smartphones. *OSDI*.

[10] Au, K. W. Y., Zhou, Y. F., Huang, Z., & Lie, D. (2012). Pscout: analyzing the android permission specification. *Proceedings of the 2012 ACM conference on Computer and communications security*. ACM.

[11] Burguera, I., Zurutuza, U., & Nadjm-Tehrani, S. (2011). Crowdroid: behavior-based malware detection system for android. *Proceedings of the 1st ACM workshop on Security and privacy in smartphones and mobile devices*. ACM.

[12] Singh, A. (2014). Forensic analysis for massive mobile applications using data mining. *The International Journal of Big Data*, *1*(3).

[13] Nadji, Y., Giffin, J., & Traynor, P. (2011). Automated remote repair for mobile malware. *Proceedings of the 27th Annual Computer Security Applications Conference*. ACM.

[14] Enck, W., Octeau, D., McDaniel, P., & Chaudhuri, S. (2011). A Study of Android Application Security. *USENIX Security Symposium*.

[15] Portokalidis, G., Homburg, P., Anagnostakis, K., & Bos, H. (2010). Paranoid Android: versatile protection for smartphones. *Proceedings of the 26th Annual Computer Security Applications Conference*. ACM.

[16] Zhou, W., Zhou, Y., Jiang, X., & Ning, P. (2012). Detecting repackaged smartphone applications in third-party android marketplaces. *Proceedings of the second ACM conference on Data and Application Security and Privacy*. ACM.

[17] Schmidt, A., Bye, R., Schmidt, H., Clausen, J., Kiraz, O., Yuksel, K. A., et al. (2009). Static analysis of executables for collaborative malware detection on android. *Communications, 2009. ICC'09. IEEE International Conference on*. IEEE.

[18] Frank, M., Dong, B., Felt, A. P., & Song, D. (2012). Mining Permission Request Patterns from Android and Facebook Applications. *ICDM*.

[19] Shabtai, A., Fledel, Y., & Elovici, Y. (2010). Automated static code analysis for classifying android applications using machine learning. *Computational Intelligence and Security (CIS), 2010 International Conference on*. IEEE.

[20] Sanz, B., Santos, I., Laorden, C., Ugarte-Pedrero, X., & Bringas, P. G. (2012). On the automatic categorisation of android applications. *Consumer Communications and Networking Conference (CCNC), 2012 IEEE*. IEEE.

[21] Zhou, W., Zhou, Y., Jiang, X., & Ning, P. (2012). Detecting repackaged smartphone applications in third-party android marketplaces. *Proceedings of the second ACM conference on Data and Application Security and Privacy*. ACM.

[22] Chen, Z., Han, F., Cao, J., Jiang, X., & Chen, S. (2013). Cloud computing-based forensic analysis for collaborative network security management system. *Tsinghua Science and Technology*, *18*(1), 40-50.

[23] Li, T., Han, F., Ding, S., & Chen, Z. (2011). LARX: Large-scale anti-phishing by retrospective data-exploring based on a cloud computing platform. *Computer Communications and Networks (ICCCN), 2011 Proceedings of 20th International Conference on*. IEEE.

[24] (2010). OpenIndiana. Recuperado de http://openindiana.org/.

[25] (2009). napp-it // webbased ZFS NAS/SAN appliance for OmniOS. Recuperado de http://www.napp-it.org/.

[26] Source, S. (2010). Spring framework.

[27] (2013). Twitter Bootstrap - Wikipedia, la enciclopedia libre. Recuperado de http://es.wikipedia.org/wiki/Twitter_Bootstrap.

[28] (2012). Safe Browsing API — Google Developers. Recuperado de https://developers.google.com/safe-browsing/?hl=es.

[29] (2009). Smali and baksmali - Google Code. Recuperado de https://code.google.com/p/smali/.

[30] (2013). readelf - GNU Binary Utilities. Recuperado de https://sourceware.org/binutils/docs/binutils/readelf.html.

[31] (2011). ded Homepage - SIIS. Recuperado de http://siis.cse.psu.edu/ded/.

[32] (2010). android-apktool - A tool for reverse engineering Android apk files. Recuperado de https://code.google.com/p/android-apktool/.

[33] (2010). androguard - Reverse engineering, Malware and goodware Recuperado de http://code.google.com/p/androguard/.

[34] (2013). secure-software-engineering/soot-infoflow-android · GitHub. Recuperado de https://github.com/secure-software-engineering/soot-infoflow-android.

[35] (2007). strace(1): trace system calls/signals - Linux man page. Recuperado de http://linux.die.net/man/1/strace.

[36] (2010). randoop - Random test generation - Google Project Hosting. Recuperado de https://code.google.com/p/randoop/.

[37] Takanen, A., Demott, J. D., & Miller, C. (2008). *Fuzzing for software security testing and quality assurance*. Artech House.

[38] Mulliner, C., & Miller, C. (2009). Fuzzing the phone in your phone. *Black Hat USA*, *25*.

[39] (2012). UI/Application Exerciser Monkey | Android Developers. Recuperado de http://developer.android.com/tools/help/monkey.html.

[40] (2012). monkeyrunner | Android Developers. Recuperado de http://developer.android.com/tools/help/monkeyrunner_concepts.html.

[41] (2009). Downloads - dex2jar - Tools to work with android ... - Google Code. Recuperado de http://code.google.com/p/dex2jar/downloads/list.

[42] Havelund, K., & Pressburger, T. (2000). Model checking java programs using java pathfinder. *International Journal on Software Tools for Technology Transfer*, *2*(4), 366-381.

[43] (2012). Intent Sniffer | iSEC Partners. Recuperado de https://www.isecpartners.com/tools/mobile-security/intent-sniffer.aspx.

[44] (2014). SafeCandy | Laboratorio i2t. Recuperado de http://www.icesi.edu.co/i2t/teleco/safecandy.php

Autores

Andres Navarro Ph.D.
Director del Grupo de Investigación i2t, Universidad Icesi. Profesor tiempo completo departamento de tecnología informática y telecomunicaciones. Doctor Ingeniero en Telecomunicación de la Universidad Politécnica de Valencia (España). Magíster en Gestión Tecnológica e Ingeniero Electrónico de la Universidad Pontificia Bolivariana. Se ha desempeñado como Ingeniero de Proyectos en Radiocomunicaciones, en Singer Products Inc. y Cia. Ltda., para empresas como CVC, Cerrejón, Telenariño y Ecopetrol. Gerente Técnico de COTEL S.A. Medellín, y de Compunet Medellín. Contacto: anavarro@icesi.edu.co

Ing. Sebastián Londoño Salcedo
Asistente de investigación del proyecto SafeCandy, Actualmente se encuentra cursando el último año de la Maestría en Investigación de la Universidad Icesi. Ingeniero de Sistemas de la misma. Contacto: slondono@icesi.edu.co

Ing. Christian Urcuqui
Auxiliar de investigación del proyecto SafeCandy, Actualmente se encuentra cursando el primer año de la Maestría en Investigación de la Universidad

Icesi. Ingeniero de Sistemas de la misma. Contacto: Christian.urcuqui@correo.icesi.edu.co

Ing. Johan Gomez
Desarrollador del proyecto SafeCandy, Actualmente trabaja en Password – Seguridad informática. Ingeniero electrónico y telecomunicaciones de la Universidad del Cauca. Contacto: Johan.gomez@password.com.co

Ing. Manuel Fuentes
Desarrollador del proyecto SafeCandy. Actualmente trabaja en Password – Seguridad informática. Ingeniero electrónico y telecomunicaciones de la Universidad del Cauca. Contacto: manuel.fuentes@password.com.co

Título: Análisis y caracterización de frameworks para detección de aplicaciones maliciosas en android

Nombre de los proponentes: Andres Navarro Cadavid, Sebastian Londoño Salcedo, Christian Urcuqui.
Filiación: Grupo de investigación i2t – Universidad Icesi
Correo electrónico: anavarro@icesi.edu.co, slondono@icesi.edu.co, Christian.urcuqui@correo.icesi.edu.co
Teléfono: +57 (2) 555 2334 Ext 8825
Dirección: Calle 18 # 122 – 135 Pance, Cali – Colombia.