


**Graduado en Ingeniería Informática**

Universidad Politécnica de Madrid

Escuela Técnica Superior de
Ingenieros Informáticos

# TRABAJO FIN DE GRADO

# Desarrollo de botnets sobre plataformas Android y posibles formas de explotación

Autor: Joel Beleña – t110383

Director: Jorge Dávila Muro

MADRID, JULIO 2018

## AGRADECIMIENTOS

A mis padres y hermanos, quienes han estado aguantando este parto que ha sido la carrera y que tanto ha durado y a mí en general.

A mis amigos, los que me han aguantado este último año prácticamente sólo hablando del trabajo de fin de grado en español, en inglés, en italiano, en polaco… Pero sobre todo, sobre todo, a Guille, mi tutor espiritual. Por él este proyecto es y tiene una forma y un sentido.

Y a quien me pueda estar dejando en el tintero, gracias, a todos.

## RESUMEN

En este trabajo se ha llevado a cabo el diseño e implementación de una *botnet* para dispositivos móviles Android. El sistema combina cifrado, esteganografía y la utilización de servicios de correo temporales, como canal de comunicación, con el fin de garantizar que en el intercambio de mensajes se de confidencialidad, autenticidad, integridad, disponibilidad y discreción. Abordando desde un punto de vista de análisis del estado actual de las *botnets* (para *Android)* y el funcionamiento de algunas de estas, sus puntos fuertes y débiles se han extraído y aplicado al diseño aquellos elementos que se podrían considerar clave en la arquitectura de una *botnet.* Además, una de las características de las que se ha dotado a este *malware*, es que pueda modificar algunos de sus comportamientos, lugares y modo de obtener la información o claves de cifrado. De todo esto se ha extraído que el uso de servicios de terceros para establecer las comunicaciones puede plantear un riesgo para la disponibilidad del sistema, en caso de que se caiga el servicio o cambie su comportamiento, pero que aun así, el uso de servicios de abiertos, unido a técnicas de esteganografía y cifrado, se presenta un escenario muy interesante para el despliegue de una *botnet* multipropósito.



## Palabras clave

Android, botnet, esteganogría, criptografía, seguridad, comunicación.



## RESUME

This project talks about the design and implementation of a botnet targeted to Android devices. The system combines encryption, steganography and using disposable email web services, as communication channel, to assure the confidentiality, authenticity, integrity, availability and stealth during the messages interchange between the botmaster and the bots. Analyzing the state of art of botnets (for Android) and the way some of them work and checking their strengths and weaknesses, we can apply this knowledge to the design of those functionalities that have been considered relevant for the architecture of a botnet. Moreover, one of the characteristics that has been provided to the malware, is the capacity of changing some of its behaviors, where and how to get the information or the value of the encryption keys. From all this, it has been concluded that despite of using third party services to establish the communication can bring some problems if they stop working or if they change their behavior, combining them with steganography technics and cryptography to the communication it is possible to achieve a very interesting way for a multipurpose botnet deployment.



## Keywords

Botnet, Android, steganography, cryptography, security, communication.

# Índice de contenido

## CONTENIDO


Agradecimientos ..... 3

Resumen ..... 4

Contenido ..... 5

1. Introducción y objetivos ..... 7

1.1. Motivación ..... 8

1.2. Objetivos ..... 8

1.3. Metodología ..... 8

1.4. Estructura del documento ..... 9

2. Estado del arte ..... 10

2.1. Estado del mercado de smartphones ..... 10

2.2. Estado de las botnets ..... 11

2.3. Botnets y malware en Android ..... 14

3. Análisis ..... 18

3.1. Requisitos ..... 18

3.2. Análisis de las soluciones planteadas ..... 23

3.2.1. Método de la comunicación de las instrucciones ..... 23

3.2.2. Seguridad en las comunicaciones ..... 26

3.2.3. Envío de informes, resultados y estadísticas ..... 27

3.3. Solución propuesta ..... 27

3.3.1. Servidor-envío ..... 28

3.3.2. Cliente ..... 33

3.3.3. Servidor-recepción ..... 34

4. Diseño ..... 35

4.1. Herramientas y tecnologías ..... 35

4.1.1. Servidor-envío ..... 35

4.1.2. Cliente ..... 35

4.1.4. Cliente-recepción ..... 36

4.2. Arquitectura del sistema ..... 36

4.2.1. Servidor-envío ..... 38

4.2.2. Cliente ..... 39

4.2.3. Servidor-recepción ..... 43

5. Despliegue, pruebas y resultados ..... 44
5.1. Resultados ..... 44
6. Conclusiones ..... 48
7. Trabajo futuro ..... 49
8. Anexo ..... 51
Anexo A – Glosario de Términos ..... 51
Anexo B - Plantilla JSON ..... 55

# Índice de figuras

Predicción de ventas de móviles a nivel mundial ..... 10
Distribución de mercado de SO móviles del 1er cuatrimestre de 2009 al 1ero de 2017 ..... 12
Proceso de creación del JSON ..... 33
Arquitectura global de la botnet ..... 37
Diagrama del script para el envío de órdenes ..... 38
Comportamiento general cliente ..... 39
Procesado de las órdenes ..... 40
Ejecución de módulo ..... 41
Shutdown ..... 42
Procesado de las peticiones de los clientes ..... 43
Imagen original. Template 1 ..... 46
Imagen original. Template2 ..... 46
Imagen con esteganografía. Template1 ..... 46
Imagen con esteganografía. Template2 ..... 46

# Índice de tablas

Algunas botnets basadas en Android ..... 17
análisis botnets ..... 18
Requisitos de Seguridad ..... 21
Comparación vías de comunicación cliente-servidor ..... 24
Comparación de servicios de terceros ..... 25
Resultado estegoanálisis en Template 1 ..... 45
Resultados estegoanálisis en Template ..... 45
Comparativa visual entre las imágenes originales y las que llevan esteganografía ..... 46
Análisis de tiempos durante la ejecución del *malware* ..... 47

# 1. INTRODUCCIÓN Y OBJETIVOS

Vivimos en un mundo altamente conectado con gran cantidad de dispositivos que están en continua comunicación y compartiendo información con todo tipo de entidades y empresas. Los intereses económicos y políticos siguen siendo el motor básico de casi cualquier actividad y esto de manera directa e indirecta ha hecho que se quiera descubrir nuevas maneras para conseguir obtener más dinero y más éxito político y social, lo que se traduce en poder y control sobre el resto de individuos.

Esta ha sido siempre la base de la delincuencia, una persona o grupo que quiere conseguir fama, poder, dinero... y que buscan el camino más rápido para conseguirlo. Mafias, *lobbies* y grupos organizados han estado tratando de buscar la manera de hacer más corto y menos arriesgado que ese camino a la fama y la riqueza, lo que ha hecho que se hayan descubierto y llevados a cabo nuevos métodos cada vez más sofisticados y originales.

Y es que en cuestión de rapidez y comodidad, nadie se quiere quedar atrás. Todo el mundo quiere tener acceso a nuevos dispositivos para que les ayuden en el día a día y así facilitarles la vida. De esta manera, rara es la persona que no lleva un teléfono inteligente en el bolsillo para estar en comunicación continua con los demás, para que le recuerde eventos, para que le diga cómo llegar al trabajo… y muchos más ejemplos.

El problema viene cuando ignorantes y despreocupados a los riesgos que todo conlleva, se descuida la protección de estos mini ordenadores abriendo las puertas a todo tipo de extraños.

Los que sí que no ignoran este hecho, son aquellas entidades de las que se hablaba al principio que aprovechan que los usuarios tienen la guardia baja unido a la falta de controles sobre estos dispositivos para poder utilizarlos a su favor, llevando a cabo actividades fraudulentas ciberataques y otro tipo de las que se hablará más adelante.

Este tipo de sistemas o redes que consisten en controlar dispositivos electrónicos (*smartphones*, ordenadores, *routers* y muchos otros dispositivos que ya se verán) y poder ordenarles llevar a cabo acciones sin el conocimiento de su dueño, se conocen como *botnets* y será el tema sobre el que tratará todo el trabajo.

Para facilitar su lectura y comprensión, se ha incluido un el en el que se explican los conceptos más técnicos que se puedan encontrar a lo largo del trabajo. Se recomienda su lectura si no se está familiarizado con la seguridad, la criptografía y el mundo del *malware.*

## 1.1. Motivación

Desde el punto de vista de una atacante siempre se están buscando nuevas brechas de seguridad y maneras innovadoras de penetrar hasta el núcleo de cada dispositivo y tenerlo bajo su control.

Desde el punto de vista de los analistas e investigadores de seguridad, hay que estudiar el *software* que es desarrollado e implementado en los terminarles, pues por culpa de fallos de diseño se quedan abiertas puertas por las que los ciberdelincuentes se pueden colar y realizar actividades dañinas. Estos fallos de diseño no sólo afectan a la programación de software, también a las arquitecturas de redes o a las configuraciones de los sistemas.

Esto genera una carrera continua en la que todos quieren estar un paso por delante, los atacantes quieren vulnerar los sistemas antes de que las brechas sean cerradas y los defensores quieren conocer cuáles son las brechas que pueden querer utilizar los atacantes, para sellarlas antes de que puedan ser explotadas.

Por eso es importante que los investigadores se pongan en la piel de los atacantes y estudien su trayectoria para conocer cuáles han sido las técnicas usadas anteriormente y finalmente, poder anticiparse a sus movimientos.

En este proyecto se va realizar el estudio de algunas *botnets* basadas en dispositivos móviles que han existido y otras que siguen existiendo, analizando cuáles pueden ser sus fallos y deficiencias y tratando de definir, diseñar y desarrollar una *botnet* que utilizando nuevas técnicas permita ejecutar el código el *botmaster* quiera.

## 1.2. Objetivos

El objetivo de este trabajo es diseñar y desarrollar una *botnet* con fines académicos basada en el protocolo IP en el que se establezca una comunicación segura ente el *botmaster* y los dispositivos infectados a través de un servicio de terceros. Como lo que se quiere es descubrir nuevas técnicas que se pueden usar, se va a dar mucho peso a aquellos mecanismos que no han sido utilizados o descubiertas hasta ahora.

Uno de los objetivos principales va a ser asegurar que todas las comunicaciones que se establezcan entre los *bots* y el *botmaster* pasen lo más desapercibidas posible, siendo este un punto determinante en la toma de decisiones a la hora del diseño y la implementación.

En definitiva, el sistema programado tendrá que satisfacer los criterios de seguridad que afectan a un sistema informático: confidencialidad, autenticidad, integridad, trazabilidad, disponibilidad y control de acceso. En el apartado de  se explicará qué es cada uno, el riesgo que pueden suponer al sistema si son comprometidos o la forma en las que se van a conseguir.

## 1.3. Metodología

Para llevar a cabo este trabajo, se van a identificar en primer lugar aquellos requisitos mínimos que se considera que una *botnet* debería cumplir, así como la manera y el grado en el que pueden afectar a la disponibilidad del sistema. Además de estos, se añadirán otros requisitos

que se va a querer que esta *botnet* implemente pensando en el uso que se vaya a hacer de ella y que resuelvan deficiencias encontradas en otros sistemas.

Para la identificación de las partes a mejorar durante la investigación del estado del arte de , se va a estudiar cómo algunas de estos *malware* se comportan, identificando posibles deficiencias que podrían presentar y proponer mejoras y cambios para desarrollar un sistema más avanzada en esos aspectos.

Para cada parte crítica encontrada se propondrán varias mejoras y, en la medida de lo posible (tiempo y esfuerzo limitados) se implementará en el sistema propuesto. En los casos que no sean posible añadir dichas mejoras se incluirán ideas y soluciones en el apartado final del documento, dedicado a exponer las posibles líneas de investigación abiertas.

## 1.4. Estructura del documento

El documento se desarrolla de la siguiente manera: en el segundo capítulo se hace una introducción a las *botnets* en general, *botnets* en *Android* y otros *malware* dentro de *Android*; en el tercero, se hará un análisis de en base a otras *botnets*, cuáles son los elementos a a tener en cuenta antes de diseñar la *botnet*. Cuando se tenga hecho el análisis y los requisitos bien definidos, se entrará a detallar el diseño de la *botnet.* Una vez terminado el tema del desarrollo, se pasará a desplegar el proyecto y estudiar si se ha conseguido cubrir las expectativas planteadas, y en caso contrario, qué se podría hacer para conseguirlo.

# 2. ESTADO DEL ARTE

En este apartado se va a analizar en primer lugar cómo se encuentra en la actualidad el mercado de los *smartphones*, entrando más en detalle en *Android*, a continuación el ecosistema de las *botnets* y por último se hablará de los dos elementos en conjunto, haciendo un análisis de cómo se encuentra el entorno actual y concluyendo con el espacio que se pretende ocupar con este trabajo.

## 2.1. Estado del mercado de *smartphones*

Actualmente, y como podemos observar en la Ilustración 1 el mercado de los smartphones (móviles inteligentes) no deja de crecer y se espera que hacia 2019 el número de dispositivos que se vendan sea de 2 billones.

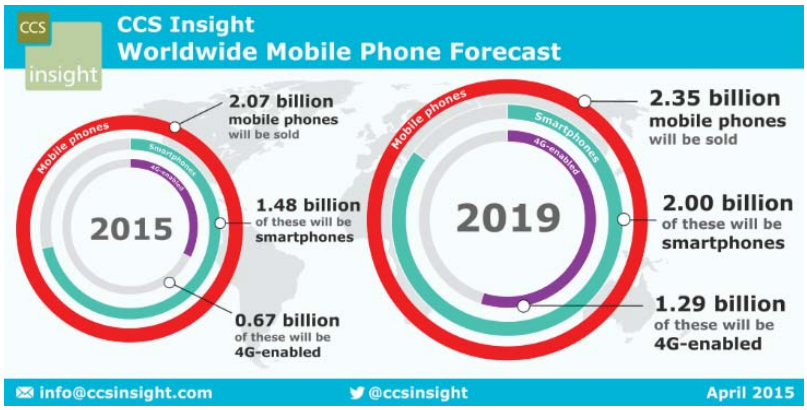


**Ilustración 1: Predicción de ventas de móviles a nivel mundial**[1]

Si se observa la Ilustración 2, se puede ver una alta superioridad por parte de dispositivos *Android* en comparación con el resto de sistemas operativos que podemos encontrar dentro de esta "nueva" generación de teléfonos móviles.

*Android* cuenta con unas características muy particulares con respecto a sus competidores. En primero lugar está basado en Linux, y por tanto cualquiera puede desarrollar su propia versión de él. Existen algunos proyectos abiertos en los que se trabaja de manera conjunta para mantener y desarrollar una alternativa a *Android*, conocida como *"CyanogenMod*". Puede ser considerado de código abierto a pesar de que va unido a una parte de *software* propietario, lo que permite que investigadores y analistas puedan buscar fallos de seguridad, reportarlos y que se puedan sacar parches que corrijan los fallos identificados por la comunidad. Teniendo en cuenta que los *smartphones* no son más que pequeños ordenadores, y no por ello algunos dejan de ser menos potentes, hacen que puedan ser utilizados para realizar tareas como las

1 http://www.ccsinsight.com/press/company-news/2183-smartphone-sales-to-peak-in-western-markets-in-2017-as-they-enter-new-phase-of-maturity

que se podrían llevar a cabo desde un portátil o un equipo de mesa. Siendo por tanto un elemento muy parecido a un PC, la perspectiva de poder tener control de dichos dispositivos, o mejor aún, de una gran cantidad de ellos, representa una actividad que puede llegar a ser muy lucrativa dentro del mundo de la ciberdelincuencia. Observando la trayectoria de otras *botnets*, los grupos cibercriminales pueden llegar a obtener grandes beneficios a partir del gobierno de una *botnet*.

## 2.2. Estado de las botnets

Como ya se ha hecho referencia en el glosario de términos, una *botnet* es un conjunto de equipos infectados (también conocidos como *zombis* o *bots*) que reciben órdenes desde un servidor de comando y control (C&C o 2C o C2).

En la primera fase de la consolidación de una botnet, el atacante por medio de diferentes técnicas consigue instalar en el terminal de la víctima un *malware* que le permite tomar control de este. Según la *botnet*, las formas de conseguirlo varían mucho. Una de las *botnets* más actuales y que ha protagonizado los mayores ataques de DDoS conocidos hasta la fecha, es "*Mirai*" [2].

2 https://en.wikipedia.org/wiki/Mirai_(malware)

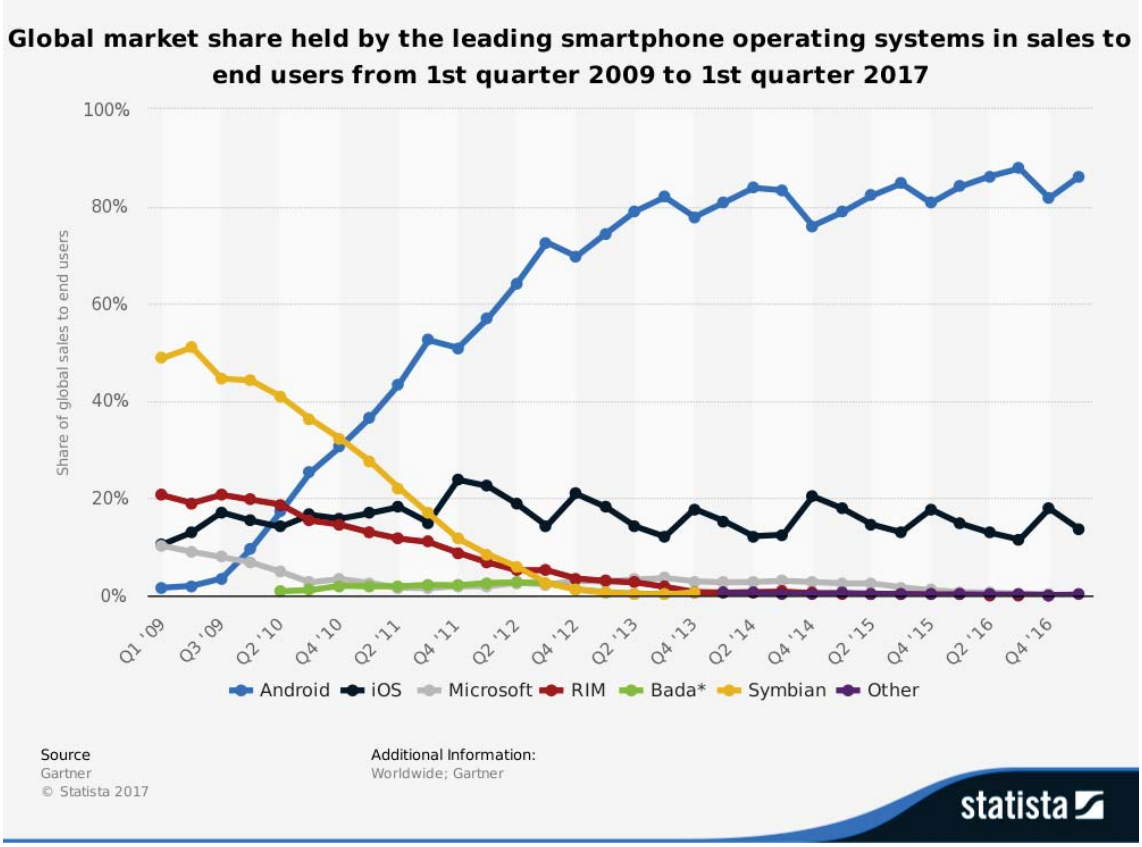


**Ilustración 2: Distribución de mercado de SO móviles del 1er cuatrimestre de 2009 al 1ero de 2017**[3]

"*Mirai*" es un *malware* que se aprovecha de vulnerabilidades y malas configuraciones en dispositivos *IoT* (*Internet of Things* – Internet de las cosas) como cámaras IP, impresoras conectadas por *WiFi*, neveras, termostatos… para infectarlos y convertirlos en *zombis* de una *botnet*. Otro medio muy popular para hacerse con el control de estos dispositivos es a través del envío de *spam,* intentar engañar a la víctima para que, o bien acceda a un enlace a una web maliciosa, o bien se descargue un fichero adjunto al correo que provoca la infección del equipo.

Una vez infectado, dependiendo del tipo de *botnet* y cómo esta esté programada, podrá llevar a cabo diferentes comportamientos.

Una característica que se usa mucho para categorizar el tipo de *botnet,* es la forma en la que lleva a cabo las comunicaciones (recibir instrucciones y enviar resultados). Dentro de esta categoría, encontramos dos modelos clásicos:

---

3 https://www.statista.com/statistics/266136/global-market-share-held-by-smartphone-operating-systems

- Modelo **cliente-servidor**: en este tipo de red los *zombis* se conectan a un servidor central (C&C) que será el que les de las órdenes y les diga cómo tienen que actuar. En el código malicioso se puede encontrar la IP (pueden ser varias) o el dominio (pueden ser varios) donde tienen que hacer las consultas de las nuevas órdenes o dónde tienen que notificar que un nuevo dispositivo se ha unido al entramado. Dentro de esta familia, puede ser que no haya sólo un servidor, sino que sean varios a los que se puedan hacer las consultas.

- Modelo **P2P**: en este tipo de arquitectura unos *bots* actúan como cliente y otros hacen las funciones de C&C. Este modelo se puede dividir en tres sub-grupos: centralizado (un servidor se actualiza periódicamente indicando en qué nodos está qué información), descentralizado-estructurado (el contenido de qué información hay y en qué dispositivo se encuentra distribuida en varios supernodos) y descentralizado-no-estructurado (no hay una lista con la localización de los recursos, cuando se quiere encontrar algo hay que preguntar a todos los nodos de la red).

Una característica interesante de las redes P2P es que los *bots*, según su naturaleza, pueden comportarse como servidores de comando y control, o como meros trabajadores que consultan y ejecutan lo que se les ordene desde los C&C.

Una idea común es creer que en este tipo de redes no hay ningún dispositivo fijo, pero en la realidad, es necesario que hay uno o varios *bots* que tengan una IP pública o fija y que sobretodo, permita la conexión desde fuera para poder establecer un canal de comunicación con los *bots* que funcionan como actuadores. [4] [5]

A partir de aquí, los *bots* o bien están configurados para realizar siempre una misma tarea, o quedan pendientes de recibir instrucciones del servidor que se le haya dicho diciéndoles qué y cuándo tienen que hacer. En algunos casos, las *botnets* están destinadas a sólo realizar una tarea concreta, en otros, son utilizadas con múltiples propósitos.

Algunas de las actividades que se pueden llevar a cabo con una *botnet* son las siguientes:

- Realizar **ataques DDoS**. Ordenando a muchos dispositivos móviles solicitar o acceder a un recurso o dirección público en internet al mismo tiempo, o enviando peticiones mal formadas que limite la capacidad del servidor.

---

4 https://www.malwaretech.com/2013/12/peer-to-peer-botnets-for-beginners.html

5 http://www.cs.ucf.edu/~czou/research/P2PBotnets-bookChapter.pdf

- Campañas de **SPAM**. Desde sólo un dispositivo se pueden enviar miles de correos en un sólo día sin que el usuario lo sepa. Puede ser usado para infectar a nuevos terminales.
- Minar **criptomonedas**. Quizás no es la mejor idea para un *smartphone* pues la cantidad de recursos que esto requiere puede ralentizar y agotar rápidamente la batería del terminal, alertando así al usuario. Pero conseguir dinero de manera directa encontrando monedas criptográficas usando los recursos de otra persona siempre es una opción para los cibercriminales.
- Robo de **información personal**. La recolección de datos como cuentas de correo, claves de acceso a cuentas bancarias, credenciales de acceso a VPNs… es una actividad de la que es muy fácil obtener beneficios, ya que existe un gran mercado de gente dispuesta a comprar esta información.
- Alterar **votaciones**. Hacer que todos los *bots* voten de forma masiva en una encuesta en internet puede servir para falsear los resultados de un referéndum haciendo a este inválido o simplemente, dando como ganador al que pueda interesar al que gobierne la red de *zombis*.

Aunque algunas de estas actividades no implican un beneficio económico directo, como sí lo son por ejemplo minar criptomonedas, las *botnets* en sí tienen un valor económico propio basado en diferentes parámetros, como su extensión (número de dispositivos infectados) o la cantidad de ancho de banda que pueden llegar a generar (si se puede hacer que llegue un gran volumen de información a un servicio, este se puede quedar sin recursos para atender a todas las peticiones y se podría llegar a conseguir un ataque de denegación de servicio - DoS). Con esto, un ciberdelincuente puede alquilar o vender la *botnet* a quien pueda estar interesado en la realizar de un ataque a un competidor, a nivel empresarial o político, llevar a cabo campañas de SPAM o alterar resultados de encuesta. [6]

## 2.3. Botnets y malware en Android

Como se ha visto en el primer apartado, el mercado de los dispositivos móviles no deja de crecer y además las actividades criminales están tomando cada vez más presencia en el mundo digital. Esto es fácil de entender si lo comparamos con las actividades que las mafias han estado realizando hasta ahora. El vertiginoso crecimiento de Internet, los usos que de él se hacen y las opciones que presenta, ha supuesto un reto para los organismos legislativos y judiciales que no han sabido afrontar. La venta ilegal de armas, el tráfico de drogas y el tráfico de personas, es un grupo de actividades que reportando grandes beneficios, representan

6 https://securelist.lat/la-economa-de-las-botnets/67377/

además un gran riesgo, implican una exposición de alguna manera o de otra en el mundo tangible, con grandes penas para aquellos que se encuentren envueltos en estos delitos. En cambio, las actividades criminales llevadas a través de internet, permiten a las mafias y a los grupos cibercriminales operar a través de la pantalla de un ordenador o un teléfono, reduciendo su exposición y gracias a la cantidad de técnicas y métodos para aumentar el anonimato, haciendo mucho más difícil su localización. [7]

Uniendo estos dos elementos se puede ver cómo prolifera la aparición y aumenta el uso de *malware* y en concreto, las *botnets* en estos terminales.

Un primer ejemplo podría ser la *botnet* a la que pertenecen los dispositivos infectados por el *malware* "*NotCompatible.C*"[8] [9]. En la tabla Algunas botnets basadas en Android se recogen algunos ejemplos, qué tipo de *botnet* son y otras características que presentan.

Como es de esperar, los grupos que operan con cualquier tipo de *malware*, siempre buscan tratar de ocultar sus programas a analistas de seguridad, antivirus y otros tipos de escáneres automáticos y especializados en la detección de actividades maliciosas, así como la utilización de métodos alternativos para la comunicación para poder alargar el tiempo de vida y dificultar el desmantelamiento de estos por medio del cierre o bloqueo de los servidores que utilizan.

En cuanto a la protección de la comunicación y del *payload*, se han seleccionado algunas técnicas interesantes para conseguirlo:

- La técnica de ofuscación del código es una práctica común, que consiste en modificar los nombres de las variables, funciones, objetos... y la ejecución de rutinas (manteniendo el resultado de la ejecución) con el fin de evadir a los antivirus y a los analistas de *malware.*
- El *rootkit "*Alureon" descargaba imágenes de servicios de internet de las que extraía ficheros de configuración (ocultos en ellas) con información de servidores de C&C disponibles. [10]
- En junio de 2017 se descubrió que un grupo de *hackers* estaban utilizando el perfil de Britney Spears en Instagram para publicar comentarios en los que codificaban

7 https://blog.lookout.com/notcompatible

8 https://info.lookout.com/rs/051-ESQ-475/images/Lookout_NotCompatibleC_Whitepaper_v2.1_10-26-2016.pdf

9 https://www.forbes.com/sites/tonybradley/2015/10/16/cybercrime-is-the-modern-day-mafia/1/

10 https://threatpost.com/alureon-rootkit-morphs-again-adds-steganography-092611/75688/

el id de una url de donde descargar una pieza de *software*, que con el fin de añadir capas extra de seguridad, su contenido se encontraba cifrado. [11]

- El uso del email también se ha empleado para la actualización del *malware* y para intercambio de información como se descubrió que "Icoscript" estaba haciendo. [12]
- El uso de servidores IRC ha sido (y sigue siendo) una práctica común en cuanto a lo que la comunicación de la *botnet* se refiere. [13]

11 https://www.welivesecurity.com/2017/06/06/turlas-watering-hole-campaign-updated-firefox-extension-abusing-instagram/

12 https://www.wired.com/2014/10/hackers-using-gmail-drafts-update-malware-steal-data/

13 https://en.wikipedia.org/wiki/Zeus_(malware)

| Nombre | Modelo *Botnet* | Fecha Descubrimiento | Número de dispositivos infectados | Estado Actual | Usos |
|---|---|---|---|---|---|
| **NotCompatible.C** | Cliente-Servidor<br>10 servidores distribuidos geográficamente | 2012 | Desconocido | Activa | Campañas de spam<br>Compra masiva de entradas en sitios como ticketea<br>Se cree que se alquilan sus servicios para diferentes usos |
| **Trojan.Android/Spy.Banker.HW**[14] | Cliente-Servidor | 6 de Febrero 2017 | Más de 2800 | Desactivada el 23 de Febrero de 2017 | Robo de credenciales de bancos |
| ***Android/Twitoor*** [15] | Cliente–Servidor.<br>Se utiliza a "*Twitter*" como servidor para dar las órdenes a los *bots* | Agosto de 2016 | Desconocido | Inactiva desde septiembre de 2016 | Instalar malware |
| **WireX** [16] | Cliente-Servidor | 2 de Agosto de 2017 | Más de 120.000 IPs únicas | Activa | Ataques DDoS |
| **Spamsoldier** [17] | Cliente-Servidor | 26 de Octubre de 2012 | Desconocido | Inactiva desde Diciembre de 2012 | SMS spam |

**Tabla 1: Algunas botnets basadas en Android**

14 https://www.welivesecurity.com/2017/02/23/released-android-malware-source-code-used-run-banking-botnet/

15 https://www.welivesecurity.com/2016/08/24/first-twitter-controlled-android-botnet-discovered/

16 https://blog.cloudflare.com/the-wirex-botnet/

17 https://blog.cloudmark.com/2012/12/16/android-trojan-used-to-create-simple-sms-spam-botnet/

# 3. ANÁLISIS

| Servicio | Característica a estudiar | Problema | Solución |
|---|---|---|---|
| **NotCompatible.C** | Uso de servidores propios | Vulnerable a una desconexión coordinada | Utilizar servicios de terceros, pudiendo añadir o modificar a necesidad |
| **TeleRAT**[18] | Montado sobre telegram | Aunque la idea es buena, es necesario tener Telegram instalado | No usar servicios que requieran registro |
| **Turla**[19] | Mandar órdenes a través de instagram | La idea es buena, usa esteganografía con texto en vez de imágenes | Ninguna. Usar estegannografía sobre imágenes |

**Tabla 2: análisis botnets**

Una vez estudiadas algunas de las técnicas que se utilizan en del mundo de las *botnets* para *Android*, y cuáles de ellas han funcionado y por qué, e incluso cuáles no han funcionado y también su porqué, se tiene una visión más amplia de posibles líneas a explotar. Se va a tratar de utilizar métodos innovadores que o bien aún no se han utilizado, o bien se están utilizado pero que aún no se han descubierto, para satisfacer los objetivos propuestos.

## 3.1. Requisitos

Cuando se analiza un sistema informático desde el punto de vista de la seguridad, hay unos requisitos que se tienen que cumplir para garantizar que nuestro ecosistema no puede ser corrompido ni desmantelado. A estos elementos los vamos a llamar **requisitos de seguridad** y son los siguientes:

- **Confidencialidad**: la información compartida sólo puede ser conocida por el emisor y el receptor.

---

[18] https://www.securityartwork.es/2018/05/07/telerat-un-nuevo-troyano-que-utiliza-telegram-como-covert-channel/

[19] https://www.welivesecurity.com/2017/06/06/turlas-watering-hole-campaign-updated-firefox-extension-abusing-instagram/

- **Integridad**: el mensaje no ha sido alterado por un tercero durante el trayecto.
- **Disponibilidad**: independientemente de los sucesos externos, el sistema debe seguir funcionando correctamente.
- **Trazabilidad**: capacidad de poder rastrear hasta el origen cualquier operación llevada a cabo.
- **Autenticidad**: poder determinar que el remitente es quien tiene dice ser y no alguien que se esté impersonando.
- **Control de acceso**: que la información sólo pueda ser modificada o accedida por aquellos que tengan permiso para ello.

Como en este caso se está desarrollando un código malicioso, se va a tener en cuenta otro criterio, que es la **discreción**. Se va a tratar esta propiedad como la capacidad del sistema de pasar desapercibido tanto a usuarios como a herramientas automáticas para detección de *malware*, como pueden ser los antivirus.

En la Tabla 3 se explica para cada punto cuáles son algunos posibles ataques que podrían afectar a la red, de qué manera repercutirían y el nivel de daño que se podría llegar a generar.

Por otro lado, cuando se habla de arquitecturas de sistemas distribuidos (aquellas en las que toman parte clientes y servidores), hay que tener en cuenta el des/acoplamiento espacial y temporal. Según la situación pueden ser beneficiosos o no.

Para el caso que se va a estudiar, se va a buscar un sistema que se consiga el máximo desacoplamiento, espacial y temporal, posible. Las razones de esto son:

- Acoplamiento espacial: implica que el cliente tenga que conocer el destino final del recurso (servidor) con el que se va a establecer la comunicación y viceversa. El uso de un intermediario que facilite la comunicación entre las partes puede proporcionarnos desacoplamiento. Si por ejemplo la comunicación se establece a una IP, esto sería acoplamiento espacial. Si se hace uso de DNS y se solicita un recurso a un nombre de dominio, se consigue el desacoplamiento espacial.
- Acoplamiento temporal: se da cuando es necesario que la comunicación se produzca en el momento que las dos partes participen de forma activa. Por ejemplo, una

llamada de teléfono implica acoplamiento temporal, pero en cambio, el envío de una carta, proporciona desacoplamiento.

| Criterio | Ataque | Consecuencia | Alcance e Impacto |
|---|---|---|---|
| **Confidencialidad** | Intercepción de mensajes | Conocimiento del contenido del *payload* | MEDIO<br>Conocer detalles e intenciones de la campaña que se está llevando a cabo |
| **Integridad** | Modificación del *payload* | Poder determinar qué van a recibir los clientes | CRÍTICO<br>Con un mensaje concreto se pueden desactivar todos los terminales, desmantelando la *botnet* |
| **Disponibilidad** | Bloquear el acceso a los servicios en los terminales | Impedir la recepción de nuevas órdenes | BAJO<br>Ya que sólo afecta a un terminal |
| | Parar el funcionamiento de uno o varios servicios | Impedir el envío de nuevas órdenes | BAJO – CRÍTICO<br>Según el número de servicios desconectados, la *botnet* podría dejar de funcionar |
| **Trazabilidad** | Tratar de enviar mensajes no legítimos para estudiar el comportamiento del sistema | Análisis de la *botnet* | MEDIO - ALTO<br>Conocer que el sistema ha sido descubierto pudiendo ser comprometido |
| **Autenticidad** | Modificación del *payload* | Desactivación de los *bots* | CRÍTICO<br>Posibilidad de desactivar todos los terminales, desmantelando la *botnet* |
| **Control de acceso** | Modificación de informes a enviar | Falsear la información recogida y estadísticas de infección | BAJO |
| **Discreción** | Descubrimiento | Análisis de la *botnet* | MEDIO - ALTO<br>El descubrimiento del sistema puede comprometerlo |

**Tabla 3: Requisitos de Seguridad**

Se considera que los dispositivos no se encuentran siempre online. Pueden experimentar fallos de conectividad, estar apagados, sin cobertura… Esto podría provocar la pérdida de mensajes si no se establecen mecanismos de control, lo que aumentaría la complejidad de la programación.

La dirección IP de los dispositivos es desconocida, así como si son o no visibles desde internet. Debido a que un *smartphone* se puede conectar a través de WIFI o GSM, en ningún momento tendrán una dirección pública contra la que iniciar una comunicación, y además al encontrarse tras un *router* (como mínimo), y que generalmente (si no siempre) éstos utilizan NAPT, los puertos con los que establecer la conexión estarán muy limitados. [20]

Por otra parte, para hacer que el sistema sea moldeable, se requiere que los *bots* puedan ejecutar cualquier instrucción que se les envíe, de esta manera, una vez que la *botnet* esté en funcionamiento, no habrá problemas en caso de querer modificar su comportamiento pues se podrá adaptar según las necesidades que puedan ir surgiendo.

El momento de ejecutar de este código tendrá que poder determinarse, ya que en algunas ocasiones se querrá que todos los clientes ejecuten el código al unísono.

Teniendo en cuenta que según el efecto que se quiera conseguir se querrá que no todos los *bots* ejecuten las órdenes, sino sólo aquellos que tengan el mismo modelo de teléfono o que se encuentren en el mismo país.

Desde el punto de vista del cliente, también se quiere que sea lo más discreto posible al usuario, haciendo el menor uso posible de los recursos posible y llamando la atención lo menos posible.

El control de errores que se produzcan en el dispositivo tiene que estar muy cuidado ya que si la aplicación deja de funcionar, no habría manera de mantener la comunicación ni de volver a levantar el proceso. Para ello hay que además establecer mecanismos en caso de que el teléfono se reinicie.

Un detalle muy importante a tener en cuenta es que como un analista va a tener acceso a esta parte del sistema, en sus manos tendrá acceso al código, las claves de des/cifrado, informes generados y puntos de comunicación cliente/servidor. Por esta razón habrá que utilizar

20 https://www.pcworld.com/article/2955112/phones/your-mobile-ip-address-its-safety-is-one-thing-its-privacy-is-another.html

técnicas de ofuscación de código y añadir varias capas de seguridad para dificultar el acceso a la información sensible y al conocimiento del funcionamiento del sistema.

A continuación, y tras el estudio del estado actual del *malware,* y los requisitos detallados, se plantearán una serie de opciones (que se han considerado innovadoras y efectivas) para garantizar (o al menos proporcionar) el máximo nivel de seguridad y robustez al sistema.

## 3.2. Análisis de las soluciones planteadas

En esta sección se van a exponer y estudiar diferentes opciones que se pueden usar para satisfacer los requisitos arriba planteados.

### 3.2.1. Método de la comunicación de las instrucciones

Debido a la complejidad que plantea, se descarta la arquitectura P2P. Así, se plantean las siguientes opciones basadas en el modelo cliente-servidor:

- **IRC**: el *bot herder* y los equipos infectados se utilizan a través de este protocolo pudiendo establecer una comunicación en tiempo real con la máquina, lo cual no siempre es una ventaja porque eso implica acoplamiento espacial y temporal.
- **Servicio web propio**: uso de un servicio web programado por el *botmaster.* Puede coincidir o no, que el mismo servicio web sirva tanto para dar las órdenes como para recibir mensajes de los *bots*.
- **Servicio web de terceros**: establecer las mismas relaciones de comunicación (envío y recepción a través de servicios web que ya existen. Redes sociales, foros, blogs...
- **Servicio oculto (*hidden service*)**: este planteamiento es el mismo que los dos anteriores, pero va sobre la red TOR, ganando en anonimato y seguridad.

Pros y contras de cada opción:

| Opción | Ventajas | Desventajas |
|---|---|---|
| IRC | • Comunicación en tiempo real entre el *bot* y el *botmaster*<br>• Más control en la información | • Tráfico no común, lo que puede llamar la atención por su rareza<br>• Acoplamiento espacial y temporal |
| Servicio web propio | • Desacoplamiento espacial y temporal<br>• Tráfico HTTP/S muy raramente es bloqueado por un *firewall* o *router* | • Se tiene que programar<br>• Costes de mantenimiento<br>• Responsabilidad legal directa |
| Servicio web de terceros | • Desacoplamiento espacial y temporal<br>• Exención de responsabilidades | • Si no proporciona API hay que analizar el modo en el que se realizan las peticiones HTTP |

| | | |
|---|---|---|
| | • Tráfico HTTP/S muy raramente es bloqueado por un *firewall* o *router*<br>• El tráfico malicioso se confunde con el tráfico legítimo de los usuarios | • Si cambia el servicio o se cae, se pierde el canal de comunicación |
| *Hidden service* | • Correctamente configurado garantiza anonimato. IP desconocida.<br>• Desacoplamiento espacial y temporal | • Se tiene que programar<br>• Costes de mantenimiento<br>• Responsabilidad legal<br>• Requiere configuración<br>• El tráfico por la red TOR es desconfiable |

**Tabla 4: Comparación vías de comunicación cliente-servidor**

Teniendo en cuenta la tabla anteriormente expuesta (Tabla 4), los pros y contras de cada opción, el tiempo de desarrollo y teniendo en cuenta el reto y lo innovador de la solución que se puede llegar a conseguir, se va a montar el sistema de comunicación sobre **servicios de terceros**.

En la siguiente tabla (Tabla 5) se presentan diferentes servicios que se pueden utilizar, de qué manera y algunas de sus características:

| Servicio | Categoría | Comprime imágenes | Requiere registro | Medidas anti bots | ¿Tiene API? | Inconvenientes |
|---|---|---|---|---|---|---|
| **Twitter** | [RS] Microblogging | Sí | Sí, pero no siempre con medidas anti *bot* | - | Sí | Limitación de caracteres |
| **Imgur** | [RS] Subida imágenes | No | Según para qué | - Genera URLs aleatorias cuando subes la imagen.<br>- Registro anti *bots* | Sí | Los comentados en las medidas anti-bots |
| **Yopmail** | [SW] Correo temporal | No | No | Rara vez para ver algún correo.<br>Para enviar correos | No | No es un servicio muy usado y puede causar sospechas en el tráfico |
| **MailCatch** | [SW] Correo temporal | No | No | No | No | No es un servicio muy usado y puede causar sospechas en el tráfico |
| **Pastebin** | [SW] Publicar texto | - | Según para qué | Sí | Sí, pero la útil es de pago | Las publicaciones pueden ser denunciadas y eliminadas |

**Tabla 5: Comparación de servicios de terceros**

### 3.2.2. Seguridad en las comunicaciones

Como ya se ha dicho que la comunicación tiene que ser segura para garantizar la confidencialidad, integridad y autenticidad de los mensajes se van a exponer qué opciones pueden ser usadas para conseguir tales fines.

En cuanto a la confidencialidad, las opciones que se plantean son:

- **Cifrado simétrico:** los *bots* compartirían la clave de des/cifrado con el *botmaster*
- **Cifrado asimétrico:** se utilizan dos claves para el cifrado y descifrado. Se calculan una clave privada y una pública asociadas de manera que con todo lo que se cifre con la pública, sólo puede ser descifrado con la privada (que obviamente se tiene que almacenar de manera segura).
- **Esteganografía:** este método se podría utilizar para ocultar el código a ejecutar, los parámetros y otros elementos necesarios a comunicar al bot en imágenes.

Para la integridad, se puede utilizar:

- ***Hashing*:** acompañar al mensaje del *hash* generado en base al contenido de este para que el *bot* pueda comprobar que no faltan partes o está corrupto.
- **Firma digital:** la firma digital consistiría en acompañar el mensaje con un *hash* (igual que en la opción anterior), pero que además va cifrado con una clave privada. Luego, con la clave pública, el mensaje y la firma, se podría comprobar que el mensaje no ha sido alterado, además de que proviene exclusivamente del *botmaster*, al estar firmado con la clave privada que sólo él conoce.

Y en cuanto a la autenticidad:

- **Firma digital:** como se ha explicado anteriormente, con esta técnica podemos conseguir garantizar que el mensaje proviene del *botmaster*, además de que este ha sido el original escrito por él.

La disponibilidad hay que entenderla desde el punto de vista del cliente y el servidor, no habiendo mecanismos que se puedan llevar a cabo desde el punto de vista del cliente, pues si desde este se bloquea el acceso a los servicios que se usan, el *botmaster* no puede hacer nada. Por otro lado, la mejor solución para garantizar la disponibilidad de la recepción de órdenes es ofrecer a los *bots* varios servicios desde donde recibir instrucciones.

La trazabilidad es un criterio difícil de satisfacer en este entorno ya que el *bot herder* no tiene presencia en los terminales y no puede más que conocer la información que estos le pasen. Una cosa que sí se podría hacer es que en caso de que algún cliente reciba un mensaje que no corresponde al formato que tiene que ser, que la firma digital no sea legítima o se detecten intentos de extraer información de la aplicación, que notifique al *botmaster* para ponerle en conocimiento de ello. Debido al bajo impacto que esto puede llegar a aplicar al sistema y las

medidas correctoras que se podrían llevar a cabo para evitarlo serían muy costosas, no se van a aplicar mecanismos para satisfacer este criterio.

En el cliente, el control de acceso a la información sí que es algo que puede llegar a ocurrir, toda la información que se puede considerar sensible, va a ser cifrada con **cifrado simétrico**.

Como se ha dicho que se quiere pasar desapercibido, una de las mejores formas de hacerlo es, si se utilizan servicios de tercero, simular el comportamiento de un usuario más. Como se ha decidido que se van a usar servicios de terceros, las opciones que se plantean son:

- Cifrar la información para que no pueda ser leído por el resto de usuarios.
- Como el hecho de publicar información cifrada genera un tráfico sospechoso e incongruente a la vista de cualquiera, se puede hacer uso de la esteganografía para que estos mensajes no sean visibles.

### 3.2.3. Envío de informes, resultados y estadísticas

Aunque es cierto que no en todos los casos sea necesario que los *bots* necesiten enviar información de vuelta o el resultado de la operación realizada, siempre puede ser útil, sobre todo para este trabajo en el que la obtención de estadísticas puede ayudar a exponer los resultados obtenidos.

Como se ha expuesto que se va a querer utilizar siempre que se pueda un servicio de terceros, se van a exponer servicios que podrían ser útiles:

- **Correos temporales:** el inconveniente de utilizar estos servicios es que algunos no permiten enviar correos, y si lo permiten, sólo es dentro de la plataforma o con la resolución de un *captcha*.
- **Servicios de paste:** aunque estos servicios pueden ser muy útiles para publicar gran cantidad de información, e incluso algunos de ellos permiten hasta asignar un título personalizado, o bien se quedan publicados a todos los usuarios o tienen medidas anti *bots* que dificultarían el uso de los mismos.
- **Redes sociales:** el problema que presentan estas plataformas principalmente es que requieren la necesidad de alta y registro de usuarios para poder publicar, con todos los inconvenientes que ello conllevaría, como por ejemplo: dejar las claves de acceso en la apk que usan los clientes, tener que darse de alta superando controles anti-bots...

## 3.3. Solución propuesta

Una vez expuestas y analizadas diferentes opciones para cada apartado, se va a proceder a detallar qué características va a tener cada módulo. Esta selección se va a basar en el conocimiento que ya se tiene tanto de tecnologías, programación y seguridad, las ventajas e inconvenientes que plantea cada posibilidad, pero sobretodo, en lo innovador de la idea y el interés generado en el autor a la hora de idear el sistema.

El sistema va a consistir en tres componentes: el servidor-envío, el cliente y el servidor-recepción.

### 3.3.1. Servidor-envío

Cuando se habla del servidor, se habla al componente que se encarga de preparar y mandar las instrucciones a los clientes.

Finalmente se ha decidido utilizar los servicios de correo temporales para el paso de las órdenes a los *bots*. En concreto, se van a utilizar las webs: “yopmail.com” y “mailcatch.com”.
Aunque estas opciones pueden generar tráfico sospechoso, por el hecho de que no son servicios muy populares, estas plataformas presentan varias ventajas:

- No es necesario activar cuentas ni dar de alta nada, así que en cualquier momento se puede ordenar que los bots esperen órdenes de una nueva cuenta, e incluso usar esto para gestionar campañas. Excepto en “yopmail.com” que para consultar el *feed* de emails hay que superar un captcha, pero una vez hecho esto, con indicarle al bot el “*hash* de validación” y el email, es suficiente.

- Al ser correos, las imágenes adjuntas no se comprimen y como se verá más adelante, dado que la cantidad de información que se quiere enviar puede ser muy grande, se van a utilizar para ocultar la información.

Como se quiere pasar desapercibido y se va a utilizar un servicio de correo como intermediario, el mensaje que se enviará a los *bots* tendrá el aspecto de un email de SPAM. Para ello se definirán diferentes plantillas basadas en correos de SPAM originales, detallando el cuerpo del mensaje y las imágenes que lo acompañarán.

Por otro lado, como se quiere que los clientes puedan ejecutar cualquier código que se les envíe, se va a crear una carpeta en la que se incluyan los distintos módulos que se querrá ejecutar según las necesidades que vayan surgiendo.

Para das las órdenes a los *bots* se va a utilizar un fichero con formato JSON que estará dividido en dividido en tres partes: restricciones, operación e información a actualizar. Para generarlo, se llevará a cabo un proceso dividido en cuatro fases:

1. Elección del *template* para el email.
2. Elección del módulo que se va a ejecutar y los parámetros que acompañan a la ejecución del código (en caso de que sean necesarios), así como el momento de ejecución.
3. Procesamiento del *payload*, adaptándolo al *template* elegido y envío a la dirección de correo de donde el cliente obtendrá las órdenes.

#### *3.3.1.1. Templates*

Se va a crear una carpeta llamada "*templates*" donde en subcarpetas se clasificarán las distintas plantillas que se podrán usar. Cada carpeta va a contener un fichero llamado "*body.txt*" y otra carpeta con el nombre "*images*" donde se almacenarán las imágenes que se incorporarán al cuerpo del mensaje.

#### *3.3.1.2. Estructura del JSON*

Las partes del JSON

- **Restricciones**: determinan si ese dispositivo debe ejecutar o no la operación.
- **Operación**: acción a llevar a cabo por parte del dispositivo.
- **Actualización**: valores de configuración que deben ser actualizados por parte del dispositivo.

Para formar el mensaje se invoca el programa y este va preguntando al usuario qué opciones quiere para su mensaje.

##### **3.3.1.2.1. Restricciones**

Como una orden es común a todos los terminales, es posible que se quiera especificar cuáles de ellos sean los que lleven a cabo la operación o la actualización de la información. Para ello, en esta sección se indicarán una serie de categorías o parámetros y los dispositivos que los cumplan o se encuentren dentro de ellas, serán quienes lleven a cabo las órdenes.

Los elementos que se van a incluir en el JSON son:

```
country: String
ids: String[]
model: String
version_sdk: String
```

##### **3.3.1.2.2. Operación**

Esta sección representa la parte más importante ya que contiene la operación que se va a llevar a cabo por parte de los dispositivos. En primer lugar, se va a diferenciar entre dos tipos de operaciones a llevar a cabo: ejecución de código compilado (código "0") o *shutdown* (código "1").

- **Ejecución de código:** Las distintas órdenes a ejecutar van a recibir el nombre de "módulo". Al igual que se hizo con las plantillas de los mensajes, se va a crear una carpeta llamada "modules" que contendrá más carpetas con el nombre del módulo a ejecutar.

El objetivo de organizarlo así es que se vayan creando nuevos módulos según vayan haciendo falta y además, se puedan reutilizar para diferentes campañas.

- **Shutdown:** La función de esta operación no requiere de parámetros, ya que el cliente al recibir la orden de desconectarse, realizará los pasos necesarios para eliminar todo rastro posible de la aplicación instalada en el teléfono.

#### 3.3.1.2.3. Actualización

Como es posible que según la situación, se quieran modificar ciertos aspectos de la configuración del *malware*, esta sección se encargará de actualizar campos concretos, como por ejemplo:

```
public_key: (String) nueva clave pública para la firma digital
emails: (String[]) dirección/es de correo de donde buscar emails
syncro_time: (int) cada cuánto comprobar si hay nuevos correos
```

En el , se presenta la plantilla usada para definir los elementos del JSON.

### *3.3.1.3. Procesar JSON y envío*

Una vez el *payload* se haya generado, pasará por varios procesos y así cumplir con los requisitos de seguridad que se han establecido.

#### 3.3.1.3.1. Generación de la firma digital

Debido a la importancia que presenta el JSON para el sistema, es importante garantizar que nadie más pueda enviar mensajes con un contenido que pueda comprometer la *botnet*. Para ello, una vez que el JSON esté construido, se va a calcular la firma digital asociada a este.

Cuando se calcula una la firma digital de un fichero, se suele aplicar una función llamada HMAC, que te calcula la firma digital en base a los siguientes elementos:

- **Mensaje (m):** JSON
- **Función de *hash* (H):** SHA-256
- **Clave privada (K) y pública (K'):** Par generado mediante RSA con una longitud de 1024 *bits* (128 *Bytes*).

El proceso de generación de la firma consiste en calcular un *hash* único utilizando SHA-256 del mensaje y cifrar dicha cadena con RSA.

```
hash = H (m);
S = SF (K, hash)
```

La clave privada es secreta y se encuentra en poder del *botmaster* y es la única que puede generar las firmas.

Una vez la firma esté generada, se juntarán ésta y el payload, separándoles por un carácter determinado para poder más tarde poder separarlas:

```
nuevo_payload = S + separador + payload
```

Con esto, lo que se va a conseguir es garantizar la integridad, ya que si alguien que quiere atacar el sistema modificando el JSON, este ya no se correspondería con la firma generada y por tanto no se ejecutaría el código. Y también se garantizará la autenticidad, ya que si se genera una firma del mismo payload, pero usando una clave privada diferente, cuando el cliente fuese a aplicar la función de verificación con la clave pública asociada a la clave privada legítima del *botmaster,* dará un error no pudiendo confirmase la legitimidad del mensaje.

#### 3.3.1.3.2. Cifrado

Como los mensajes que se van a usar son abiertos, puede darse el caso que alguien intercepte la comunicación y quiera conocer el contenido del mensaje.

Para evitar eso, el mensaje va a ser cifrado usando una función de cifrado simétrico (EF) al `nuevo_payload`, usando el algoritmo AES con una clave de 256 bits (K).

```
encrypted_message = EF (nuevo_payload, K)
```

#### 3.3.1.3.3. Ocultación del mensaje

Una vez que ya se tiene el *payload* firmado y cifrado, si se enviase directamente en el cuerpo del mensaje del email, analizadores automáticos o un usuario cualquiera podría ver a simple vista que no es correo ordinario y llamar la atención, atrayendo a analistas que podrían comprometer la integridad del sistema.

Para ello, el contenido de *payload* se ocultará en las imágenes de la plantilla del correo de SPAM elegida usando esteganografía. En cada imagen se codificará el máximo que esta pueda permitir hasta que se haya procesado todo el JSON.

La técnica esteganográfica que se va a utilizar para ocultar el mensaje se conoce como Bit Menos Significativo, en inglés, *Least Significant Bit* (**LSB**) que consiste en sustituir el último bit de cada uno de los colores de un píxel. Es decir, un píxel tiene tres canales (como mínimo), uno para el color rojo, otro para el verde y otro para el azul y cada color se representa con un *byte* que equivale a 8 *bits;* si se sustituye el último *bit* de cada color, se podría ocultar un carácter (representado con un *byte*, 8 *bits*) en 3 píxeles, dado que en total se tendría:

```
3 colores * 3 píxeles = 9 colores (9 bytes)
```

Y sobraría un *bit* que se usará para almacenar el inicio del siguiente carácter.

Ejemplo. Se va a ocultar el carácter “a”, cuya representación en ASCII es (0 1 1 0 0 0 0 1), en los siguientes tres píxeles:

| RED | GREEN | BLUE | |
|---|---|---|---|
| 1 0 0 0 1 0 1 1 | 0 1 0 0 1 0 0 0 | 0 1 0 1 0 0 1 0 | |
| 0 1 0 1 1 1 1 1 | 1 0 0 1 1 0 1 1 | 1 0 0 1 0 0 1 1 | |
| 0 1 0 1 1 0 0 1 | 1 1 0 1 1 0 1 0 | 1 0 1 0 0 1 1 0 | |

Aplicando la técnica esteganográfica **LSB**, se obtendría el siguiente resultado:

| RED | GREEN | BLUE | |
|---|---|---|---|
| 1 0 0 0 1 0 1 **0** | 0 1 0 0 1 0 0 **1** | 0 1 0 1 0 0 1 **1** | |
| 0 1 0 1 1 1 1 **0** | 1 0 0 1 1 0 1 **0** | 1 0 0 1 0 0 1 **0** | |
| 0 1 0 1 1 0 0 **0** | 1 1 0 1 1 0 1 **1** | 1 0 1 0 0 1 1 0 | |

Como se puede observar, los colores apenas varían, siendo aún mucho más imperceptible cuando se trata de una imagen compuesta por una gran cantidad de píxeles.

Así, con esta estrategia se va a conseguir una capa extra de confidencialidad y de discreción.

Una vez que el *payload* esté escondido en las imágenes, se construirá un email “*Multipart/mixed*” siguiendo las especificaciones para ello en el que se añadirán las imágenes como ficheros embebidos para que aparezcan como elementos del HTML que compone el mensaje y se enviará al correo.

En la siguiente ilustración Proceso de creación del mensaje se representa de manera gráfica el proceso de transformación del JSON antes de ser enviado (sin incluir la parte de ocultación):

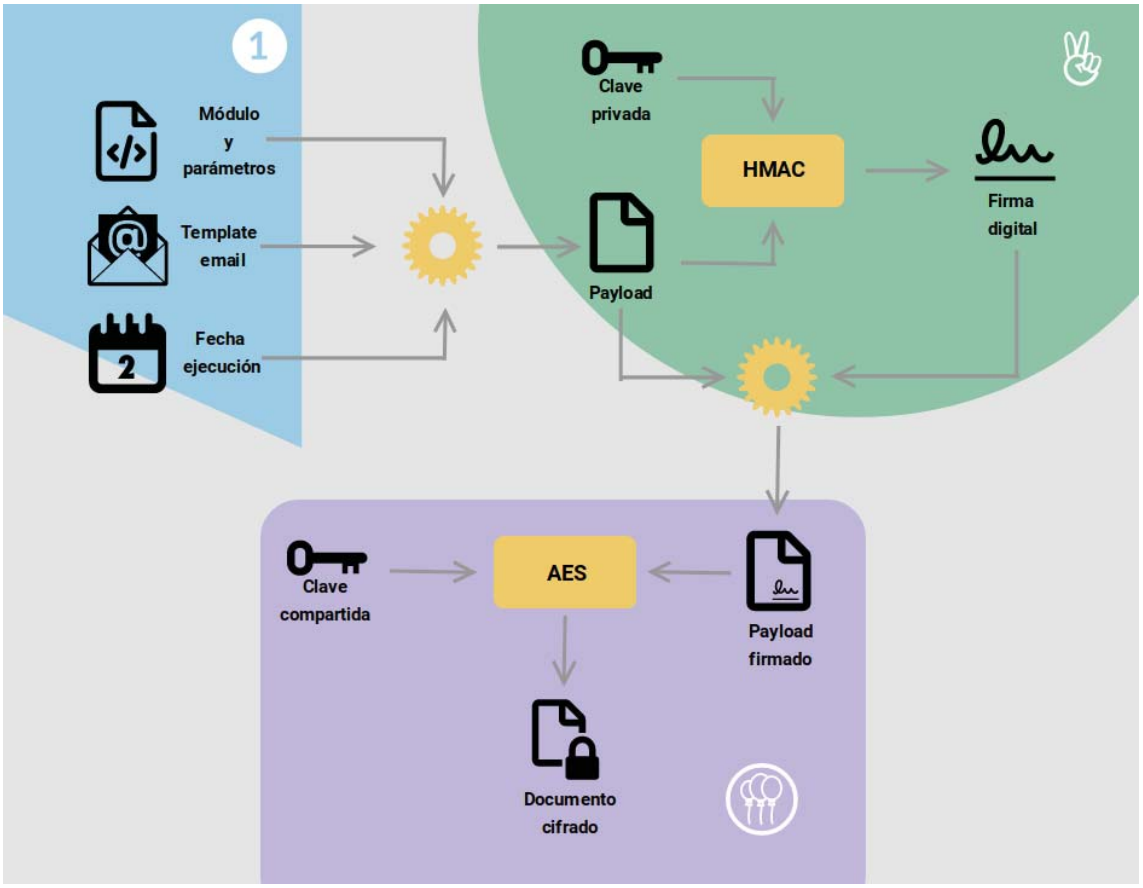


**Ilustración 3: Proceso de creación del JSON**

### 3.3.2. Cliente

La aplicación instalada en el teléfono de la víctima va a llevar por defecto una dirección de correo de donde consultarán las instrucciones a llevar a cabo. De forma periódica, comprobarán si ha llegado un nuevo email a la bandeja de entrada de cada servicio para los que estará programada.

Al encontrar correos va a realizar el proceso inverso que se ha aplicado para ocultar el mensaje, extrayendo así la operación y el módulo (en caso de que lleve) a ejecutar. Para evitar que cada vez que se envíe un módulo se tenga que hacer todo el proceso de carga, se va a ir realizando un control de versiones, almacenando exclusivamente la última versión en memoria.

Cuando se inicie por primera vez la aplicación, se va a generar un informe con la información de ese dispositivo que se quedará almacenado en memoria hasta que se pueda enviar al servidor. Además, cada vez que se lleve a cabo la ejecución de un módulo que requiera de la elaboración de un informe o envío de información se hará lo mismo, almacenarlo hasta poder enviarlo. Como estos ficheros van a contener información que dará pistas sobre el comportamiento del *malware,* se utilizará un cifrado simétrico para que sea más difícil conocer su contenido. Esta clave de cifrado va a ser única para cada dispositivo y se generará de manera aleatoria durante la primera ejecución. Tanto esta clave, como la que se usará para

descifrar el mensaje que trae las instrucciones, se van a almacenar de manera segura, utilizando funciones de ofuscado de código que obliguen a quien pueda tratar de averiguarlas a tener que ejecutar la aplicación y extraerlas en tiempo de ejecución de la aplicación.

No sólo esta información, sino también el propio código va a ser ofuscado utilizando herramientas específicas para ello que se dedican a sustituir el nombre de las variables, funciones y los objetos por nombres triviales dificultando la tarea de análisis estático del código.

Para reducir la superficie de exposición al infectado, además de carecer de interfaz, el icono de la app se va a ocultar del menú principal.

Por último, se va a incluir una tarea que ejecute la aplicación cuando el teléfono sea reiniciado.

### 3.3.3. Servidor-recepción

Una vez estudiadas las opciones planteadas, no se ha encontrado una solución entre ellas que cumpliese todos los requisitos al 100%. Por ello, y dado que este estudio es de carácter académico, se va a desarrollar un servicio que combina la plataforma de “pastebin.com” y la de “justpaste.it” tratando de ser lo más fiel posible a la realidad.

Este servicio va a cumplir las siguientes características:

- Va a estar expuesto a internet.
- Se puede personalizar la url con la que se accederá al contenido del *paste*.
- No requiere de registros ni medidas anti-bots, más que el uso de cookies.

Una licencia que también va a ser auto-otorgada es que al estar bajo control del servidor, se va a acceder directamente a la información enviada por los *bots*.

# 4. DISEÑO

## 4.1. Herramientas y tecnologías

### 4.1.1. Servidor-envío

Para la programación de la parte que prepara las órdenes y las envía al servicio de correo se va a utilizar el lenguaje de programación "*Python*". Los motivos son la facilidad del lenguaje y la soltura a la hora de programar con él.

Se va a realizar una *script* que interactuará con el usuario por medio de una CLI (*Comand Line Interface*) para ir guiándole en la elección de los elementos involucrados en la construcción del JSON (módulo, template, parámetros, restricciones y momento de ejecución).

Para la parte de generación de la firma digital y del cifrado del documento, se va a utilizar la herramienta "Open SSL" debido a la versatilidad que ofrece y que al ser ampliamente utilizada, se puede encontrar fácilmente la manera en la que realizar la acción de descifrado en la parte cliente.

En cuanto a la parte de esteganografía, se ha estado buscando una librería que funcionase igual tanto en "*Python*" como en "Java" (que como se dirá más adelante, es el lenguaje elegido para la programación de la aplicación). Esto es necesario porque el algoritmo para ocultar y para extraer tiene que ser el mismo en ambos extremos. Como no se ha encontrado ninguna herramienta que satisficiese este requisito, se ha decidido implementar un algoritmo propio de esteganografía basado en LSB tanto para "*Python*", como para "Java*".*

### 4.1.2. Cliente

Como ya se ha adelantado, "Java" va a ser el lenguaje elegido para la programación del cliente. Este es el lenguaje nativo que utilizan los dispositivos Android y la API oficial para el desarrollo de apps.

Este desarrollo se va a llevar a cabo con el IDE "Android Studio" ya que por una lado ya se tienen conocimientos de cómo utilizarlo y por otro este proporciona una serie de herramientas y facilidades que ayudan mucho a la hora de programar aplicaciones para este sistema operativo.

Como los servicios de correo que se van a usar para la comunicación no cuentan con una API para consulta y obtención de los correos, se va a realizar ingeniería inversa a las aplicaciones web con la ayuda de "BurpSuite" y un navegador web y así conocer los *endpoints* donde se tiene que hacer cada consulta, qué consultas hay que hacer, el orden en el que hacerlas, si utilizan GET o POST y los parámetros que habrá que enviar en cada petición.

Para las funciones de cifrado y descifrado, se ha basado la solución en un código encontrado en internet que planteaba la solución para el mismo problema planteado (compatibilidad en "Java" para la librería "Open SSL").

En cuanto a la esteganografía, ya se ha comentado que la implementación será propia para garantizar la compatibilidad entre los lenguajes.

Por último, para la simulación durante el desarrollo, se ha utilizado el entorno virtual proporcionado por “Genymotion” haciendo uso de la API 21 de “Android” por la naturaleza de algunos componentes que requieren que esta sea la versión mínima a usar.

#### 4.1.4. Cliente-recepción

Como en este caso no se tiene limitación por parte del sistema operativo a la hora de elegir el lenguaje de programación, se va a volver al uso de “Python” para el desarrollo del servidor.

Se va a utilizar un *framework* que se llama “Flask” que permite el desarrollo y despliegue de aplicaciones web de manera rápida y sencilla, que en combinación con “Jinja2” facilita la manera de generar páginas web dinámicas con el uso de templates.

La base de datos a utilizar va a ser SQLite por la naturaleza del proyecto. Se necesita algo rápido y ligero, ya que la capacidad que se va a soportar no es muy grande.

## 4.2. Arquitectura del sistema

El esquema aquí presentado se aplica al caso concreto que se ha desarrollado. Al no haber encontrado un servicio de terceros en tiempo, se ha implementado la siguiente solución en la que el atacante tiene acceso directo a la base de datos donde se almacenan los informes de los infectados, lo que no es un escenario que no se pueda dar.

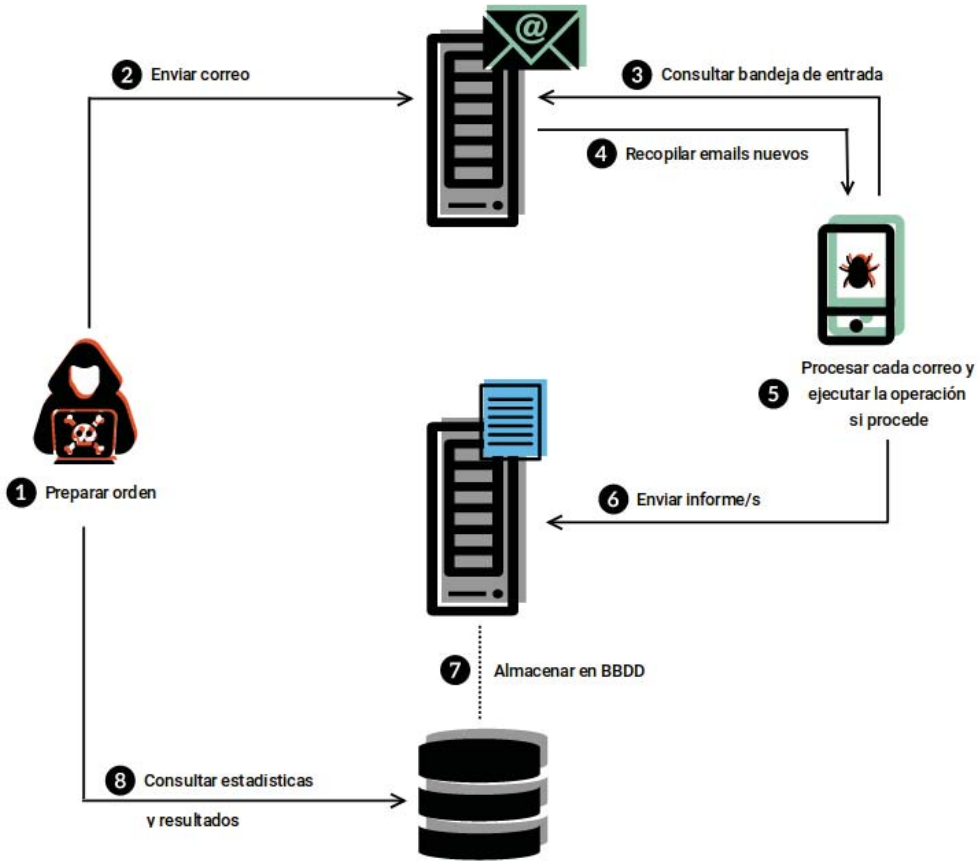


**Ilustración 4: Arquitectura global de la botnet**

### 4.2.1. Servidor-envío

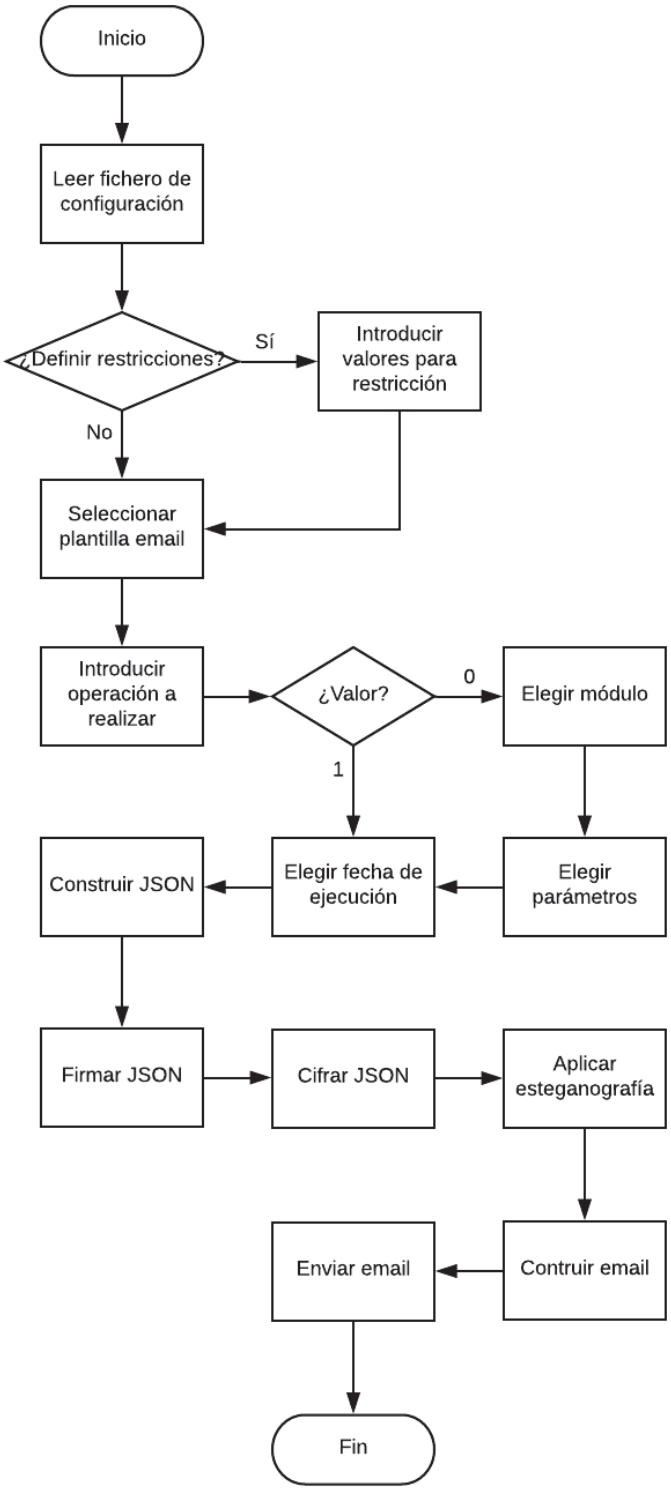


**Ilustración 5: Diagrama del script para el envío de órdenes**

### 4.2.2. Cliente

A continuación se muestran los diagramas de flujo que afectan a la aplicación que se ejecuta en el cliente.

En la Ilustración7: Procesado de las órdenes se muestra el diagrama al que se hace referencia en la Ilustración 6: Comportamiento general cliente cuando se menciona la subrutina “Buscar nuevas órdenes”.

En el caso de la Ilustración 8: ejecución de módulo e Ilustración 9: shutdown, son las subrutinas que en la Ilustración7: Procesado de las órdenes se llaman “Preparar módulo” y “*Shutdown*” respectivamente.

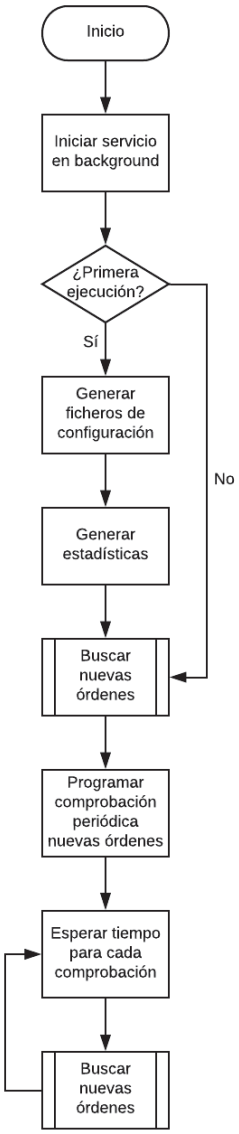


**Ilustración 6: Comportamiento general cliente**

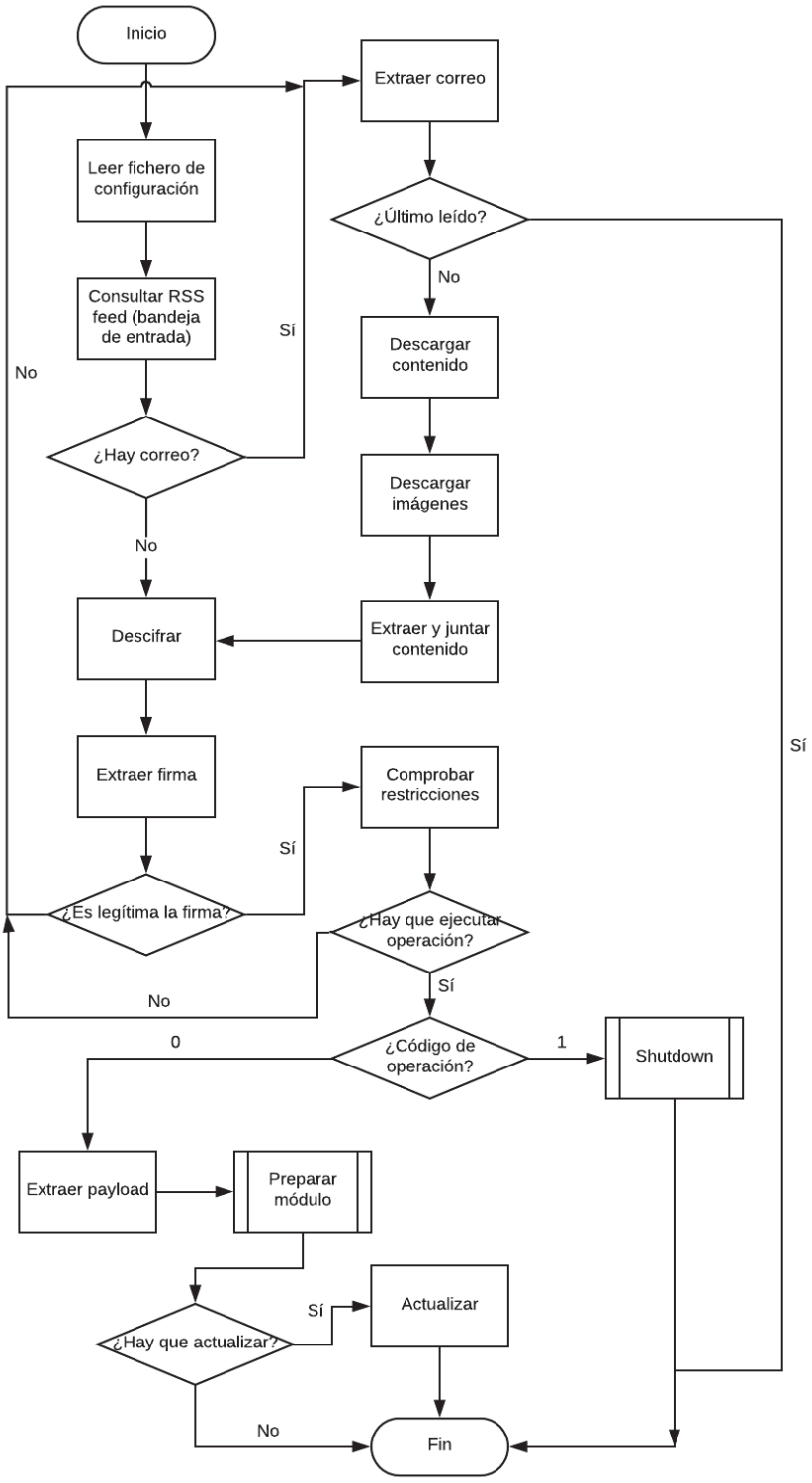


**Ilustración 7: Procesado de las órdenes**

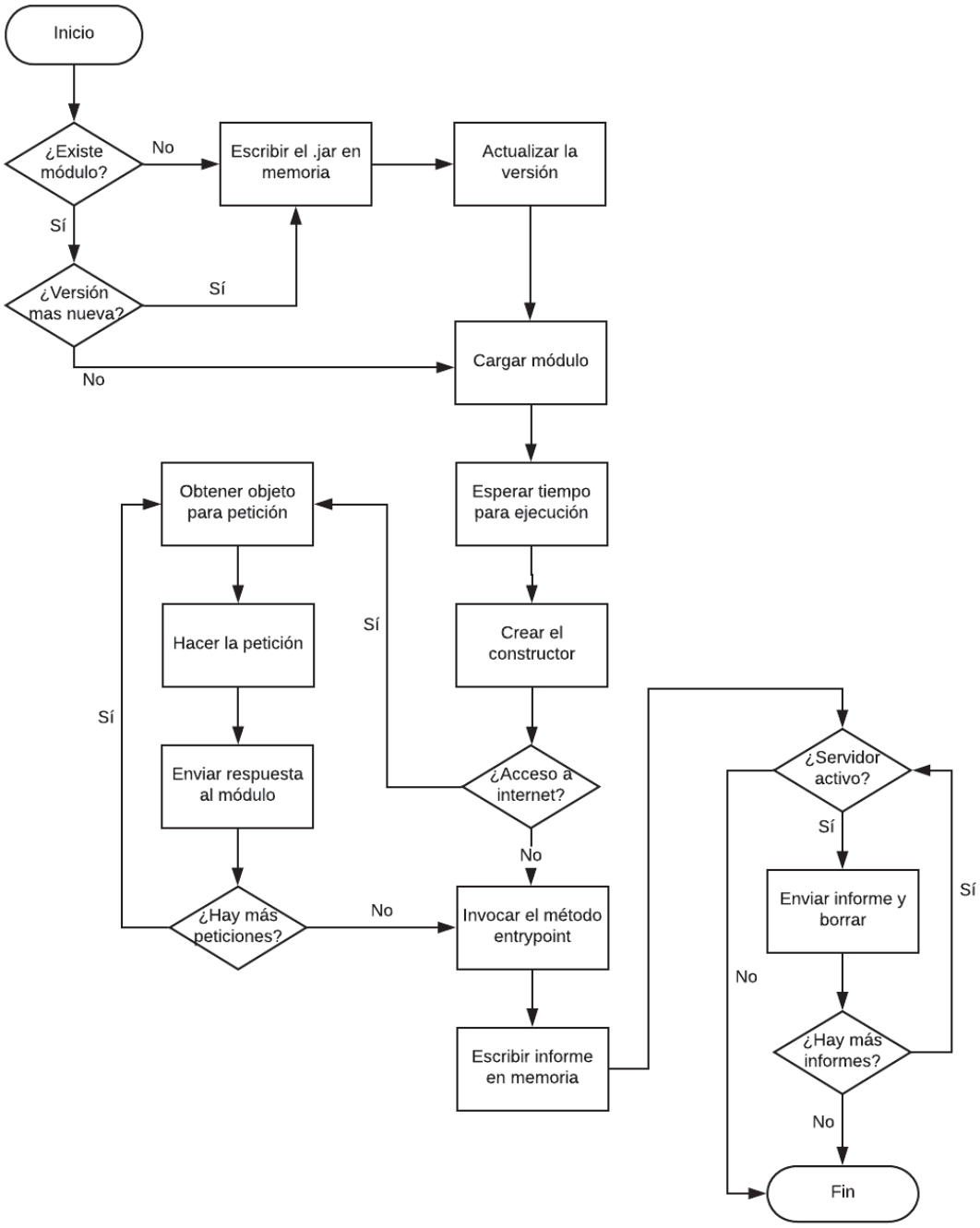


**Ilustración 8: Ejecución de módulo**

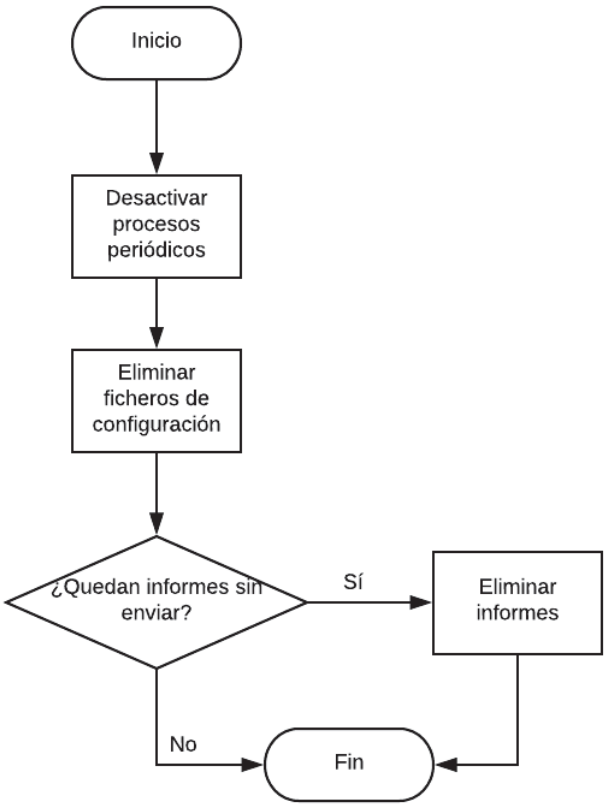


**Ilustración 9: Shutdown**

### 4.2.3. Servidor-recepción

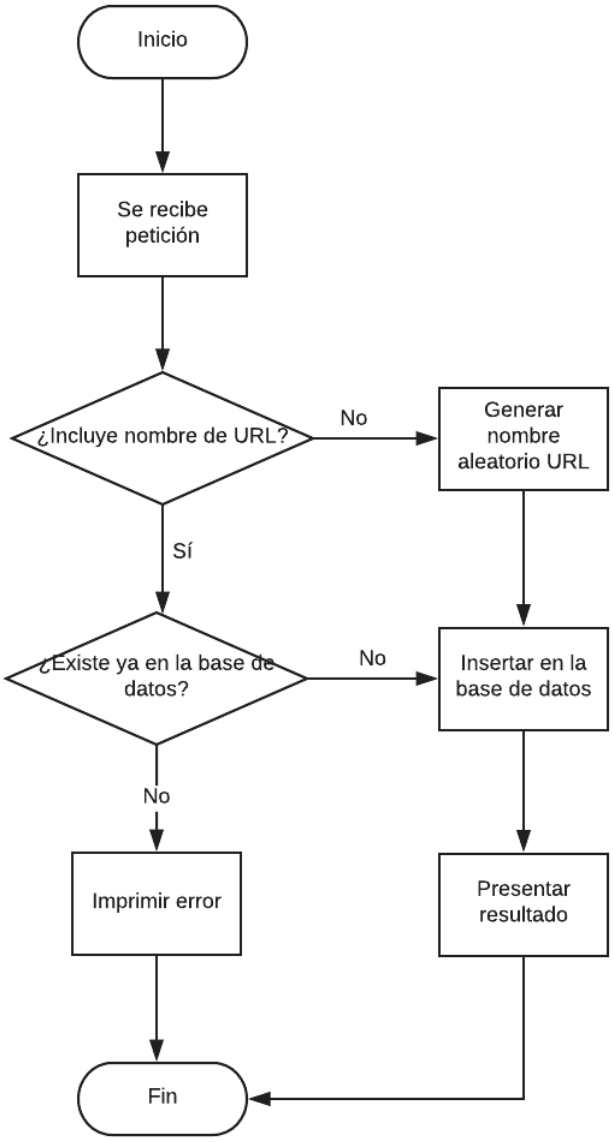


**Ilustración 10: Procesado de las peticiones de los clientes**

# 5. DESPLIEGUE, PRUEBAS Y RESULTADOS

Las pruebas que se van a llevar a cabo van a ser para dos dispositivos Android, uno de ellos se ejecutará en una máquina virtual y el otro en un terminal físico. Habrá que tener en cuenta la versión de los dispositivos, pues la aplicación está desarrollada para terminales con una versión de igual o superior a la 21.

El servidor de recepción va a estar corriendo en un VPS propio.

Y el servidor de envío se va a ejecutar desde el equipo local.

Para llevar a cabo las pruebas se van a seguir los siguientes pasos:

1. Poner en marcha el servidor de recepción de las órdenes.
2. Levantar las máquinas virtuales e instalar el *malware* tanto en ella, como en el teléfono a usar.
3. Mandar una orden usando el módulo de test que consiste en un PING-PONG enfocado a la recolección de estadísticas.

Se van a ejecutar dos pruebas en las que se utilizarán dos *templates* distintos para los correos a enviar.

El tamaño del fichero a esconder mediante esteganografía es de aproximadamente 30 kilobytes.

## 5.1. Resultados

Tras realizar las pruebas, se han obtenido los siguientes resultados.

La plantilla 1 incluye tres imágenes en las que se ha repartido el *payload*. Siguiendo el algoritmo, en cada una se ha almacenado el máximo posible hasta almacenar todo el contenido del mensaje.

Y en la plantilla 2, sólo se ha hecho uso de una imagen, ya que se considera suficiente (dado su tamaño) para almacenar la información esperada.

Tras aplicar las técnicas de esteganografía se ha obtenido lo siguiente:

- En el primer caso el tamaño de cada imagen ha aumentado de 9,2 Kb a 30,75 Kb, de 10,3 Kb a 22,87 Kb y de 80,9Kb a 781,5Kb. Hasta un total de 734,8 Kb.

- En la segunda plantilla, el tamaño de la imagen ha aumentado considerablemente, pasando de 72,6 Kbytes a 360,7 Kbytes.

Este aumento se lo podríamos atribuir principalmente al cambio de algoritmo de compresión de las imágenes, ya que originalmente tenían una extensión ".jpg" y tras el procesamiento es ".png".

En ambos casos, se ha conseguido satisfacer parcialmente el requisito de discreción, pues como se puede observar en la Tabla 8 no se aprencian diferencias a simple vista entre las imágenes originales y las modificadas. Y cuando se dice parcialmente es porque la técnica que se ha utilizado para la ocultación es vulnerable a estegoanálisis.

Las imágenes han sido analizadas con la herramienta "*Virtual Steganography Laboratory*" que permite realizar tareas de ocultación, revelación y análisis de la técnica LSB (la técnica aplicada para este proyecto). [21]

Esta aplicación ha revelado la siguiente información:

| | Tamaño (kB) | Tamaño info oculta (real) | Tamaño info oculta (detectado) | % del tamaño real | % del tamaño detectado |
|---|---|---|---|---|---|
| **Imagen 1 con esteg.** | 30802 | 11135 | 10120 | 36.15 | 32.86 |
| **Imagen 2 con esteg.** | 22825 | 8397 | 5138 | 36.79 | 22.51 |
| **Imagen 3 con esteg.** | 779936 | 9228 | 6686 | 1.18 | 0.86 |

**Tabla 6: Resultado estegoanálisis en Template 1**

| | Tamaño (kB) | Tamaño info oculta (real) | Tamaño info oculta (detectado) | % del tamaño real | % del tamaño detectado |
|---|---|---|---|---|---|
| **Imagen original** | 72547 | 0 | 2700.89 | 0 | 3.72 |
| **Imagen con esteg.** | 360665 | 30016 | 30038.04 | 8.322404 | 8.3285153 |

**Tabla 7: Resultados estegoanálisis en Template**

21 http://vsl.sourceforge.net/


**Ilustración 11: Imagen original. Template 1**


**Ilustración 12: Imagen original. Template2**


**Ilustración 13: Imagen con esteganografía. Template1**


**Ilustración 14: Imagen con esteganografía. Template2**

**Tabla 8: Comparativa visual entre las imágenes originales y las que llevan esteganografía**

Como se puede observar en la Tabla 6 y la Tabla 7 el análisis llevado a cabo por el programa ha detectado con gran precisión la cantidad de información oculta en cada una de las imágenes. Además en la segunda tabla se ha incluido el análisis realizado a la imagen original para excluir que los resultados no han sido casuales.

Esto confirma lo que ya se conocía, y es que el la técnica de LSB no funciona frente a análisis específicos para la detección de esteganografía aplicada a imágenes.

También se puede ver en la Tabla 9 ha realizado un análisis del tiempo que tarda cada dispositivo en llevar a cabo los hitos más importantes en la obtención, procesado y ejecución de las órdenes.

En este se puede ver que hay determinadas partes que emplean mucho para desempeñar la tarea. En comparación con el tiempo que se emplea para la construcción del JSON, representa un valor muy alto. Teniendo en cuenta que los dispositivos móviles no tienen la potencia computacional que pueda tener un ordenador, es normal que se puedan dar estas diferencias, pero en el caso del "ensamblado y ejecución del contenido" se están empleando más de tres minutos en el peor de los casos. Sería interesante poder estudiar la manera de reducir este tiempo para acelerar el proceso.

Con respecto a la disponibilidad, ha resultado que algunos de los servicios de correo utilizados han presentado fallos de conexión, caídas en el servicio y comportamiento no esperado por parte de la web.

Por último, durante las pruebas ha sucedido en más de una ocasión que la aplicación se ha detenido y no por excepciones directas de la aplicación, sino por excepciones generadas desde el sistema. Para conocer el origen del problema, habría que mejorar los casos de prueba y poder acotar las posibles razones. Y aunque no se ha podido determinar el origen del fallo, se cree que muy probablemente pueda ser debido a una mala gestión de los recursos, generando fallos de memoria a nivel del sistema.

| | **Template 1** | | **Template 2** | |
|---|---|---|---|---|
| | **OnePlus5** | **MV** | **OnePlus5** | **MV** |
| **Inicio proceso** | 0'' | 0'' | 0'' | 0'' |
| **Obtención lista de correos RSS Feed** | + 240ms | + 240ms | + 135ms | + 398ms |
| **Obtención de Email** | + 2ms | + 2 ms | + 8ms | + 6ms |
| **Extracción y descarga de las imágenes** | + 136ms | + 206ms | + 156ms | + 183ms |
| **Extracción información imagen 1** | + 8'' 293ms | + 15'' 707ms | + 3' 9'' 64ms | + 1' 47'' 627ms |
| **Extracción información imagen 2** | + 15'' 696ms | + 14'' 161ms | - | - |
| **Extracción información imagen 3** | + 14'' 139ms | + 10'' 298ms | - | - |
| **Ensamblado contenido y ejecución del módulo** | + 1' 28'' 65ms | + 1'22'' 657ms | + 3' 8'' 732ms | + 1' 46'' 605ms |
| **Publicación informes y estadísticas** | + 952ms | + 60ms | + 480ms | + 282ms |

**Tabla 9: Análisis de tiempos durante la ejecución del *malware***

# 6. CONCLUSIONES

Tras haber llevado a cabo las pruebas de ejecución de la *botnet* en un entorno controlado, se puede determinar que se han satisfecho los requisitos propuestos en su gran mayoría. A excepción del algoritmo esteganográfico que podría ser mejorable y los problemas que ocasionalmente podría provocar la caída de la aplicación, se ha conseguido garantizar aquellos criterios de seguridad establecidos en el punto 3.1.

Mediante las técnicas aplicadas se ha conseguido garantizar una comunicación segura garantizando la autenticidad, integridad, confidencialidad y discreción.

Debido al problema de conexión que se ha producido en algunos de los servidores, esto plantea algunos inconvenientes que pueden ser críticos para el funcionamiento del sistema. Aunque se están usando tres servicios de correo para enviar las órdenes, si por algún casual algún otro cambiase su modo de funcionamiento o también dejase de funcionar, la disponibilidad del servicio se podría ver comprometida gravemente.

Con todos estos resultados, se concluye que la *botnet* satisface la mayoría de los criterios con óptimo resultado, pero otros no, haciendo que no se pueda depositar una total confianza en su correcto funcionamiento, siendo interesante buscar nuevos métodos de intercambio de las órdenes. Por otro lado, se ha conseguido presentar un sistema muy potente en cuanto a las posibilidades que plantea, ya que es posible ejecutar cualquier tipo de código Java que se le ordene, y adaptar su funcionamiento según las necesidades, pudiendo salvar los posibles problemas de caída de los servicios o mitigar el posible compromiso del sistema, modificando además el comportamiento de algunas de sus funciones según las necesidades que vayan surgiendo.

# 7. TRABAJO FUTURO

Durante la realización de este trabajo, se ha tenido que tomar muchas decisiones sobre módulos y mecanismos, que por razones de tiempo y/o falta del conocimiento necesario, han implicaban tener que descartarlos para esta fase y dejarlos para posibles líneas futuras.

La evolución y cambios que podría experimentar este trabajo se podría clasificar en dos categorías:

- **Internos:** mecanismos y técnicas para mejorar la comunicación y qué se intercambia…
- **Externos:** aplicando cambios a la arquitectura física del sistema. Con quién se producen las comunicaciones y otros agentes que puedan afectar a la manera en que se lleva a cabo la infección

A **nivel interno** y a la vista de los resultados, hay dos cambios muy claros que se podrían llevar a cabo:

- Mejorar el algoritmo o cambiar la técnica de esteganografía y así evadir técnicas de estegoanálisis.
- Para conseguir reducir el tiempo de ejecución de algunas rutinas, se podrían utilizar mecanismos de programación concurrente para conseguir realizar tareas de manera paralela y reducir la carga en el hilo principal.

A **nivel externo**, hay dos ideas que pueda merecer la pena estudiar y desarrollar:

- Un hecho que no se ha tenido en cuenta en este trabajo porque se ha considerado que la aplicación ya se encuentra instalada en el dispositivo es la manera de infección. En un primer lugar se tenía una idea que tuvo que ser descartada por su complejidad. El planteamiento consistía en utilizar una aplicación extra que funcionase de *downloader* de la aplicación final y de vector de infección de otros dispositivos. La manera de actuar sería:
    1. Buscar redes públicas en el alcance y levantar un punto de acceso (PA) con el mismo nombre que una de las redes encontradas.
    2. Cuando los demás dispositivos se conecten a ese PA y quieran acceder a una página web, serían redirigidos a un portal cautivo donde por medio de ingeniería social se les animaría a instalar la aplicación (*downloader*).

3. La aplicación recientemente instalada descargaría por un lado la aplicación de la que se ha estado hablando en este trabajo y por otro se quedaría instalada en el teléfono tratando de infectar más dispositivos.
4. En caso de ser desinstalada, los infectados ignorarían que otra aplicación ha sido instalada, teniendo la sensación que están limpios.

- Otra idea es establecer mecanismos para detectar posibles intentos de analizar el funcionamiento de la *botnet* por parte de un dispositivo. Si esto ocurriese y se quisiese expulsar a un individuo, se podría proceder con el siguiente proceso:
    1. Añadir un procedimiento en el que los *bots* generen una cadena aleatoria que se usará como clave única y específica para comunicarse con ese *bot*.
    2. Cuando se envía el primer informe, cifrada con la clave pública del servidor se envía también esta clave.
    3. Al mismo tiempo, como el *botmaster* conocer el ID de cada dispositivo infectado, puede enviar correos específicos a cada *bot* (excepto al que se quiere expulsar), cifrados con su clave personal, en el que se especifique la actualización de la clave de cifrado genérica.

Además, sería necesario seguir enviando órdenes cifradas con la clave original pues es la única manera de que los nuevos infectados puedan leer la primera orden recibida.

Aunque este mecanismo puede ser un poco lento y no es escalable, podría ser una solución que en combinación con la línea de trabajo anteriormente mencionada podría cambiar la clave de cifrado genérica para cada futura infección entrando a modificar el instalador (.apk) de la aplicación maliciosa.

- Por último, debido a los problemas de servicio, se podrían usar otros métodos que se han ido descubriendo durante la realización del trabajo. Debido a que el servicio de imágenes “imgur“ no cambia el algoritmo de compresión ni produce provoca pérdidas en la imagen, los colores se mantienen pudiendo seguir utilizando la técnica de esteganografía para garantizar la discreción. Esto, se podría combinar con “Twitter” o cualquier otra red social (habría que estudiar pros y contras de cada opción).

# 8. ANEXO

## Anexo A – Glosario de Términos

- **Código malicioso, *malware* o código dañino**: estos términos se refieren a aquellos programas o aplicaciones que realizan acciones maliciosas y/o sin el consentimiento/conocimiento del usuario. Cada autor los utiliza de una manera diferente, en este trabajo se utilizarán de manera indistinta.
- **Servidor**: lo entenderemos como la máquina, equipo o dispositivo que aloja el software o programas que atienden diferentes peticiones de los usuarios y actúan en consecuencia. Por ejemplo, un servidor puede ser utilizado para alojar ficheros que pueden ser descargados, o también, como es en el caso en el que se va a utilizar el término, puede enviar información o dar respuestas cuando una máquina o usuario hace una consulta. Un servidor puede ser ejecutado en casi cualquier dispositivo, cualquier PC o un móvil mismo pueden ejecutar el software de un servidor en el que almacenar archivos.
- **Comando y control (C&C o 2C)**: Servidor con el que los dispositivos infectados se comunican, del que reciben las órdenes y al que mandan los información.
- **Bot, zombie, equipo infectado o esclavo**: es el nombre que reciben aquellos dispositivos (en el caso de este proyecto, *smartphones* con sistema operativo *Android*) que sin el conocimiento del usuario han sido infectados por un *malware* que lo pone al servicio de un C&C, del que reciben órdenes.
- **Botnet**: Red de zombies que reciben órdenes de un servidor C&C.
- **Botmaster o bot herder (pastor de bots)**: persona o grupo que gobierna y dirigue las acciones de una *botnet*.
- ***Dos (Denial of Service* – Denegación de Servicio)**: este ataque consiste en impedir el correcto funcionamiento de un servicio, bien ralentizándolo o bien haciendo que el servidor responsable se caiga.
- ***DDoS* (*Distributed Denial of Service* - Denegación de Servicio Distribuida)**: traducido como "denegación de servicio distribuida", este tipo de ataque se basa en la utilización de varios dispositivos de forma simultánea para realizar un DoS.
- ***Firewall***: *software* que corre en un equipo que monitoriza las conexiones entrantes y salientes, pudiendo permitir o bloquear el paso de estas en base a unas reglas en función del protocolo, el puerto y la IP de origen y destino.

- **Código abierto**: cuando se dice que un *software* o programa es de código abierto, se está haciendo referencia a aquellas implementaciones de las cuales su código y programación son públicas y cualquiera puede utilizar y modificar adaptándolo a sus necesidades.

- ***Software* propietario**: se conoce con este nombre al *software* que una compañía o particular ha desarrollado, manteniendo el código de éste en privado, e incluso no permitiendo la utilización de técnicas de *reversing* para conocer su funcionamiento interno o la modificación de su comportamiento con otro que el fin que no sea para el que fue programado, pudiéndose incluso emprender acciones legales contra el que haga un uso indebido de este tipo de programas.

- **SPAM**: nombre que se utiliza para referirse al envío masivo de correos basura. El fin de estos correos es variable, va desde enviar publicidad, hasta distribución de malware.

- **IoT (Intenert of Things – Internet de las Cosas)**: siempre que se hable de aquellos dispositivos o, literalmente dicho, cosas del día a día que se conectan a internet para ofrecernos nuevos servicios o funcionalidades que puedan enriquecer nuestra experiencia con ese producto. Véase por ejemplo: luces que podemos regular desde nuestros equipos, cámaras de vigilancia que se puedan ver desde un móvil, tostadoras a las que se les pueda enviar un mensaje para que se active a una determinada hora o una cafetera que pueda tener listo el café cuando el usuario se lo indique. La poca atención que se le ha prestado a la securización de estos dispositivos, hace de ellos un gran atractivo para los ciberdelincuentes.

- **Criptomonedas**: profundizar en este término se escapa completamente del alcance de este trabajo, pero para dar una idea al lectos, son unas monedas basadas en una fórmula matemática. Realizando cálculos y por medio de muchos intentos se puede llegar a dar con una solución que satisfaga al modelo, obteniendo así una moneda de las que se estén minando. Hay varios modelos, como por ejemplo el “Bitcoin”, “Monero” o “Etherium”.

- **Payload:** cuando se hablar de malware, se refiere a la pieza de código que realiza la actividad maliciosa. Si se trata de un entorno en el que se realiza un intercambio de información entre dos o varios dispositivos, es la parte del mensaje resultante de eliminar las cabeceras y aquellos elementos que se usan para indicar elementos como de dónde viene, a quién va dirigido u otros detalles del protocolo. Es importante conocer los dos usos de este concepto, pues será usado en ambos contextos.

- ***Rootkit:*** es una familia de *malware* que consiste en un conjunto de herramientas que proporcionan a un posible atacante el acceso privilegiado a una máquina y que se

mantiene oculto al modificar o corromper el comportamiento del sistema operativo de aplicaciones concretas. Si por ejemplo hay una herramienta en el equipo que se encarga de listar los procesos y programas que se están ejecutando en él, este código malicioso podría alterar los elementos listados para ocultar su presencia.

- **IRC (*Internet Relay Chat*):** es un protocolo utilizado para comunicarse usando texto exclusivamente basado sobre un modelo cliente/servidor.
- **TOR (*The Onion Router*):** la red tor es una red montada sobre internet cuyo objetivo es garantizar el anonimato y la privacidad de los usuarios enviando la información cifrada desde el origen hasta el destino y pasando por diferentes nodos para que no se pueda relacionar el origen inicial ni el destino final. Estos nodos son los propios usuarios que utilizan la red.
- **DNS (*Domain Name Server*):** protocolo que se utiliza en internet para conocer la IP de un servidor asociado a un nombre de dominio. Podemos entender los DNS como una tabla en la que se referencia a qué IP está vinculada un dominio.
- ***Hash:*** es una cadena alfanumérica que es siempre de la misma longitud (cada algoritmo puede tener una longitud distinta), única e irreversible asociada a un fichero. Dicho de otra manera, cuando a un fichero se le aplica una función de *hash*, el valor devuelto es (o tiene que ser) único para ese fichero, tiene una lonfigud y teniendo el *hash,* no se puede conocer el contenido del fichero o el valor del que proviene.
- **Cifrado simétrico**: usando una clave única y compartida entre las partes se transforma el contenido de un fichero o texto usando un algoritmo o función de cifrado simétrico (p.e.: 3DES, AES o Blowfish) de manera que sólo se pueda obtener el fichero original usando el mismo algoritmo y clave.
- **Cifrado asimétrico**: el contenido de un fichero o texto se procesa usando un algoritmo de cifrado asimétrico (p.e.: RSA) usando una clave pública y su contenido sólo se puede revelar usando el mismo algoritmo de descifrado y la clave privada complementaria a la usada para el cifrado. La clave pública recibe este nombre porque puede ser abiertamente conocida, pero la privada tiene que ser almacenada de forma segura.
- **Esteganografía:** es la ciencia que se dedica al estudio y a la aplicación de técnicas para ocultar un información u objetos en otros pasando desapercibido. La técnica más clásica consiste en ocultar mensajes en imágenes, pero puede ser en un otro texto, audios, vídeos… e incluso formatos no digitales.
- **Estegoanálisis:** ciencia que estudia si se han aplicado técnicas de esteganografía a a un fichero.

- **JSON (*JavaScript Object Notation*):** es un estándar que define el formato en texto plano de información estructurada de manera jerárquica, organizando los datos en modo clave: valor.
- **HMAC (*Hash-based Message Authentication Code*):** es una función que combina la generación de un *hash* de un fichero y su cifrado usando una clave privada, que permita comprobar la autenticidad del mensaje con la clave pública complementaria más adelante.
- ***Downloader:*** es un *malware* cuya única función es la de descargar la pieza de código dañino que será la que lleve a cabo las acciones maliciosas.

## Anexo B - Plantilla JSON

```
{
    "restrictions":{
    "country":"${country}",
    "ids": ["${ids}"],
    "model": "${model}",
    "version_sdk": "${version_sdk}"
    },
    "operation":{
    "0":{
        "${module}":{
        "version": "${version}",
        "payload": "${payload}",
        "parameters": "${parameters}",
        "requiresInternet": "${requiresInternet}",
        "when": "${when}"
        }
    }
    },
    "_comments": "esta configuracion la podirmaos renombrar como
sharedPreferencesElements y lo que pongamos aqui lo podemos reescribir en
el SHP del movil",
    "new_configuration":{
    "public_key": "${public_key}",
    "emails": "${emails}",
    "syncro_time": "${refresh_time}"
    }
}
```

Este documento esta firmado por


| | |
|---|---|
| **Firmante** | CN=tfgm.fi.upm.es, OU=CCFI, O=Facultad de Informatica - UPM, C=ES |
| **Fecha/Hora** | Fri Jul 06 18:35:18 CEST 2018 |
| **Emisor del Certificado** | EMAILADDRESS=camanager@fi.upm.es, CN=CA Facultad de Informatica, O=Facultad de Informatica - UPM, C=ES |
| **Numero de Serie** | 630 |
| **Metodo** | urn:adobe.com:Adobe.PPKLite:adbe.pkcs7.sha1 (Adobe Signature) |