



Universidad
Carlos III de Madrid

Departamento de Informática

Análisis de ofuscación en Apps mediante casos de prueba

Trabajo Fin de Grado

Autor: Mario Herreros Díaz
Titulación: Grado en Ingeniería Informática
Tutor: Guillermo Suárez de Tangil

Leganés, septiembre de 2013

AGRADECIMIENTOS

Muchos son los nombres que han depositado su confianza en mí, numerosos los apellidos que me han transmitido su sapiencia y su influencia, e incontables los sobrenombres que me han proporcionado su apoyo durante todo este tiempo.

A todas las personas que se esconden tras estos nombres, apellidos y apodos, sinceramente, muchas gracias.

RESUMEN

La fuerte expansión de los dispositivos móviles dentro de la sociedad actual ha significado una nueva revolución en el mundo de las comunicaciones. Con la aparición de los nuevos terminales inteligentes (*smartphones*) y las tabletas, una persona puede llevar en su mano un dispositivo con prestaciones similares a las de un ordenador personal.

Estos dispositivos también comparten inconvenientes que presentan las computadoras convencionales, como son los problemas relacionados con la seguridad. Debido a la gran expansión y acogida de estos dispositivos por parte de la sociedad, los atacantes han modificado sus pautas de ataque y han desarrollado nuevas técnicas para adaptarse a este tipo de sistemas.

Muchos tipos de código malicioso utilizan técnicas de ofuscación para llevar a cabo sus ataques. La ofuscación es el acto de encubrir información con el propósito de que ésta sea confusa y difícil de leer. Con el desarrollo de este TFG se pretende estudiar el comportamiento de este tipo de *malware* que hace uso de artefactos ofuscados y que afecta a dispositivos móviles que emplean el sistema operativo Android.

El proyecto desarrollado se centra en detectar y analizar artefactos ofuscados, presentes en las aplicaciones Android, con el fin de identificar si dicho artefacto oculta alguna funcionalidad software o algún parámetro que intervenga en alguna funcionalidad, en cuyo caso se estudia si la naturaleza de dicha funcionalidad es maliciosa.

Se ha diseñado una técnica basada en *repackaging*, en la que se realiza un estudio automático de los principales elementos ofuscados, mediante un análisis estático del código descompilado de la aplicación, combinándolo con un análisis dinámico con el fin de comprobar su comportamiento en tiempo de ejecución. Utilizando esto como base, se ha desarrollado un sistema automatizado de casos de prueba para determinar, según los resultados obtenidos, si el artefacto ofuscado contiene un *payload* malicioso o no.

Palabras clave:

análisis estático, análisis dinámico, Android, dispositivo móvil, *malware*, ofuscación, *payload*, *repackaging*.

ABSTRACT

The strong growth of mobile devices inside the current society has carried a new revolution in the world of communications. With the emergence of new smartphones and tablets every person can carry in his pocket a device with similar performance to that of a personal computer.

These devices also share problems can be found in a conventional computer, such as security-related problems. Due to the expansion and reception of these devices by people, the attackers have modified his guidelines of assault, and have developed new techniques to accommodate these types of systems.

The purpose of this final project is to study the behavior of malware on mobile devices with Android operating system. A lot of malicious software use obfuscation techniques to carry out attacks. Obfuscation is the act of hiding information so that the data are confusing and difficult to read.

The study focuses on analyzing obfuscated objects present in Android applications in order to study their behavior and to carry out detections of malware.

It has been designed a technique based on repackaging, embodying an automatic study obfuscated elements. It combines static analysis of source code decompiled with a dynamic analysis to study the behavior of the running application.

Using this as a basis, it has developed an automated test cases system to determine, according to the results, if the obfuscated object contains a malicious payload or not.

Key words:

static analysis, dynamic analysis, Android, mobile device, malware, obfuscation, payload, repackaging.

ÍNDICE GENERAL

Capítulo 1: Introducción.....	15
1.1 Motivación	16
1.2 Objetivos	17
1.3 Estructura del documento	18
1.4 Acrónimos.....	19
1.5 Glosario de términos	20
Capítulo 2: Análisis	22
2.1 Definición del sistema.....	23
2.1.1 Alcance del sistema.....	23
2.1.2 Estándares y Normas.....	23
2.2 Estado del arte.....	24
2.2.1 Difusión del sistema operativo Android.....	24
2.2.2 Android: arquitectura y características.....	26
2.2.3 Malware en Android.....	31
2.2.4 Ingeniería inversa aplicada sobre malware de Android	33
2.2.5 Análisis dinámico de aplicaciones Android.	34
2.3 Casos de uso.....	37
2.3.1 Diagrama de casos de uso	37
2.3.2 Especificación de actores del sistema	38
2.3.3 Especificación de casos de uso del sistema.....	38
2.4 Requisitos del sistema.....	41
2.4.1 Estructura del requisito.....	41
2.4.2 Requisitos de usuario	42
2.4.3 Requisitos software	55
2.5 Estudio previo	77
2.5.1 Caso de prueba 1: AnserverBot.....	77
2.5.2 Caso de prueba 2: DroidKungFu.....	81
2.5.3 Caso de prueba 3: GingerMaster	86
2.5.4 Caso de prueba 4: DroidKungFu.....	90
2.5.5 Caso de prueba 5: Pjapps	94
Capítulo 3: Diseño	100

3.1 Arquitectura del sistema	101
3.1.1 Análisis de ofuscación.....	102
3.1.2 Análisis de comportamiento.....	103
3.1.3 Detección de malware ofuscado.....	103
3.2 Diagrama de clases	106
3.2.1 Clases: Análisis de ofuscación	107
3.2.2 Clases: Análisis de comportamiento	110
3.2.3 Clases: Detección de malware ofuscado	111
Capítulo 4: Implementación	114
4.1 Decompilador.....	117
4.2 Analizador estático	117
4.2.1 Analizador de cadenas de texto	117
4.2.2 Analizador de incertidumbre	118
4.2.3 Analizador de ficheros presentes en el directorio <i>assets</i>	119
4.2.4 Analizador de sentencias condicionales	119
4.2.5 Analizador de llamadas a la API <i>crypto</i>	120
4.3 Test Case Maker	120
4.3.1 Caso de prueba: Cadena de texto ofuscada	120
4.3.2 Caso de prueba: Fichero <i>assets</i>	121
4.3.3 Caso de prueba: Condicionales	121
4.3.4 Caso de prueba: Llamadas a la librería <i>crypto</i>	121
4.4 Compilador/ <i>Repackaging</i>	122
4.5 Analizador dinámico.....	123
4.5.1 Emulación de dispositivos Android	123
4.5.2 Analizador de comportamiento	123
4.5.3 Tester Automático	124
4.5.4 Análisis inicial y algoritmo de optimización de casos de prueba	126
4.6 Detección	127
Capítulo 5: Evaluación.....	128
5.1 Estudio del comportamiento	129
5.2 Resultados de detección.....	130
5.3 Discusión	135
Capítulo 6: Conclusiones	138

6.1 Conclusiones generales.....	139
6.2 Líneas futuras.....	139
6.2.1 Integración con mercados de aplicaciones Android.....	139
6.2.2 Integración con motor antivirus	140
6.2.3 Adaptación a nuevas técnicas ofuscadas maliciosas	140
6.2.4 Incorporación de técnicas basadas en Data Mining	140
Capítulo 7: Gestión del proyecto.....	141
7.1 Planificación del trabajo	142
7.1.1 Planificación inicial.....	142
7.1.2 Desarrollo real del proyecto	145
7.2 Medios técnicos empleados para el proyecto	148
7.2.1 Herramientas hardware	148
7.2.2 Herramientas software.....	148
7.3 Análisis económico del proyecto	149
7.3.1 Metodología de estimación de costes.....	149
7.3.2 Presupuesto inicial.....	149
7.3.3 Presupuesto para el cliente	152
7.3.4 Coste final y análisis de desviación.....	153
Anexo I: Bibliografía.....	158
Anexo II: Evaluación: Resultados de comportamiento	162

ÍNDICE DE TABLAS

Tabla 1: Actores del Sistema: A1.....	38
Tabla 2: Caso de uso: CU-01.....	39
Tabla 3: Caso de uso: CU-02.....	40
Tabla 4: Caso de uso: CU-03.....	41
Tabla 5: RU01. Aplicaciones Android.	42
Tabla 6: RU02. Análisis de código ofuscado.	42
Tabla 7: RU03. Análisis dinámico automático.....	42
Tabla 8: RU04. Análisis estático automático.	43
Tabla 9: RU05. Generación automática de caso de prueba.....	43
Tabla 10: RU06. Fichero de resultados.	43
Tabla 11: RU07. Análisis estático: cadenas ofuscadas.....	44
Tabla 12: RU08. Análisis dinámico: cadenas ofuscadas.....	44
Tabla 13: RU09. Casos de prueba: cadenas ofuscadas.....	44
Tabla 14: RU10. Análisis estático: assets.....	45
Tabla 15: RU11. Análisis dinámico: assets.....	45
Tabla 16: RU12. Casos de prueba: assets.....	45
Tabla 17: RU13. Tiempo de ejecución de análisis dinámico por defecto	46
Tabla 18: RU14. Selección del tiempo de ejecución de análisis dinámico.	46
Tabla 19: RU15. Análisis estático: condicionales inversos.....	46
Tabla 20: RU16. Análisis dinámico: condicionales inversos.....	47
Tabla 21: RU17. Caso de prueba: condicionales inversos.	47
Tabla 22: RU18. Análisis estático: condicionales a true.	47
Tabla 23: RU19. Análisis dinámico: condicionales a true.	48
Tabla 24: RU20. Caso de prueba: condicionales a true.....	48
Tabla 25: RU21. Análisis estático: condicionales a false.....	48
Tabla 26: RU22. Análisis dinámico: condicionales a false.	49
Tabla 27: RU23. Caso de prueba: condicionales a false.	49
Tabla 28: RU24. Análisis estático: condicionales aleatorios.	49
Tabla 29: RU25. Análisis dinámico: condicionales aleatorios.....	50
Tabla 30: RU26. Caso de prueba: condicionales aleatorios.	50

Tabla 31: RU27. Análisis estático: llamadas a API criptográfica.	50
Tabla 32: RU28. Análisis dinámico: llamadas a API criptográfica.	51
Tabla 33: RU29. Caso de prueba: llamadas a API criptográfica.	51
Tabla 34: RU30. Tester automático de interfaz.	51
Tabla 35: RU31. Optimización del número de casos de prueba.	52
Tabla 36: RU32. Definición de métrica para la detección de <i>malware</i>	52
Tabla 37: Matriz de trazabilidad: CU-RU.	54
Tabla 38: RSF01. Identificación de cadenas ofuscadas.	55
Tabla 39: RSF02. Mutación de cadenas ofuscadas.	55
Tabla 40: RSF03. Identificación de ficheros assets.	55
Tabla 41: RSF04. Mutación de ficheros assets.	56
Tabla 42: RSF05. Descompilación de la aplicación.	56
Tabla 43: RSF06. Compilación de la aplicación.	56
Tabla 44: RSF07. Caso de prueba de la aplicación sin modificar.	57
Tabla 45: RSF08. Caso de prueba: modificación string.	57
Tabla 46: RSF09. Caso de prueba: modificación fichero assets.	57
Tabla 47: RSF10. Creación AVD.	58
Tabla 48: RSF11. Eliminación AVD.	58
Tabla 49: RSF12. Ejecución caso de prueba sobre DroidBox.	58
Tabla 50: RSF13. Contador de operaciones de lecturas.	59
Tabla 51: RSF14. Contador de operaciones de escritura.	59
Tabla 52: RSF15. Contador de conexiones entrantes.	59
Tabla 53: RSF16. Contador de conexiones salientes.	60
Tabla 54: RSF17. Contador de conexiones abiertas.	60
Tabla 55: RSF18. Contador de fugas de información.	60
Tabla 56: RSF19. Tiempo medio: lecturas en disco.	61
Tabla 57: RSF20. Tiempo medio: escrituras en disco.	61
Tabla 58: RSF21. Contador de operaciones criptográficas.	61
Tabla 59: RSF22. Contador de operaciones totales.	62
Tabla 60: RSF23. Generación de fichero de resultados.	62
Tabla 61: RSF24. Identificación de sentencias condicionales.	62

Tabla 62: RSF25. Modificación de sentencias condicionales: al inverso.	63
Tabla 63: RSF26. Caso de prueba: inversión de sentencias condicionales.	63
Tabla 64: RSF27. Modificación de sentencias condicionales: true.	63
Tabla 65: RSF28. Caso de prueba: true.	64
Tabla 66: RSF29. Modificación de sentencias condicionales: false.	64
Tabla 67: RSF30. Caso de prueba: false.	64
Tabla 68: RSF31. Modificación de sentencias condicionales: aleatorio.	65
Tabla 69: RSF32. Caso de prueba: condición aleatoria.	65
Tabla 70: RSF33. Identificación de llamadas a APIs criptográficas.	65
Tabla 71: RSF34. Modificación de llamadas a APIs criptográficas.	66
Tabla 72: RSF35. Caso de prueba: modificación de llamada criptográfica.	66
Tabla 73: RSF36. Generación de un tester automático.	66
Tabla 74: RSF37. Contador UDP.	67
Tabla 75: RSF38. Contador TCP.	67
Tabla 76: RSF39. Contador DNS.	67
Tabla 77: RSF40. Contador de URLs cifradas.	68
Tabla 78: RSF41. Contador de URLs en claro.	68
Tabla 79: RSF42. Contador de HTTP POST.	68
Tabla 80: RSF43. Contador de HTTP GET.	69
Tabla 81: RSF44. Contador de POST cifrado.	69
Tabla 82: RSF45. Contador de POST en claro.	69
Tabla 83: RSF46. Contador de GET cifrado.	70
Tabla 84: RSF47. Contador de GET en claro.	70
Tabla 85: RSF48. Contador de operaciones <i>dexclassloader</i>	70
Tabla 86: RSF49. Contador de servicios iniciados.	71
Tabla 87: RSF50. Mecanismo de optimización del número de casos de prueba.	71
Tabla 88: RSF51. Detección de <i>malware</i>	71
Tabla 89: RSNF01. Santoku.	72
Tabla 90: RSNF02. APKTool	72
Tabla 91: RSNF03. Androguard.	72
Tabla 92: RSNF04. DroidBox.	73

Tabla 93: RSNF05. Tiempo de ejecución por defecto.	73
Tabla 94: RSNF06. Fichero de resultados: hoja de cálculo.	73
Tabla 95: RSNF07. Tiempo de ejecución análisis dinámico elegible.	74
Tabla 96: RSNF08. Tcpdump.	74
Tabla 97: Matriz de trazabilidad: RU-RS.	76
Tabla 98: TC1-AnserverBot. Carpeta <i>assets</i> : ofuscación de ficheros.	80
Tabla 99: TC2-DroidKungFu. Cadena ofuscada.	84
Tabla 100: TC3-GingerMaster. Ofuscación imagen asset.	88
Tabla 101: TC4-DroidKungFu. Ofuscación <i>exploit assets</i>	92
Tabla 102: TC5-Pjapps. Dirección servidor C&C ofuscada.	97
Tabla 103: Clase Fichero asset.	108
Tabla 104: Clase Condicional.	108
Tabla 105: Clase Llamada API crypto.	108
Tabla 106: Clase Cadena ofuscada.	108
Tabla 107: Clase Caso de prueba.	109
Tabla 108: Clase Análisis estático.	109
Tabla 109: Clase Compilador/Decompilador.	109
Tabla 110: Clase Análisis dinámico.	110
Tabla 111: Clase Tester automático UI.	110
Tabla 112: Clase Detección.	113
Tabla 113: Planificación inicial.	143
Tabla 114: Desarrollo real del proyecto.	145
Tabla 115: Análisis de desviaciones en la planificación.	147
Tabla 116: Desviación del presupuesto.	156

ÍNDICE DE ILUSTRACIONES

Ilustración 1: Cuota de mercado mundial de smartphones: 2011/2012	24
Ilustración 2: Top 5 SSOO: Smartphones (1Q 2013).....	25
Ilustración 3: Top 5 SSOO: Tablets (1Q 2013).....	25
Ilustración 4: Android 1.5 Cupcake.....	26
Ilustración 5: Android 1.6 Donut.....	27
Ilustración 6: Android 2.0/2.1 Éclair.....	27
Ilustración 7: Android 2.2 Froyo.....	27
Ilustración 8: Android 2.3.x Gingerbread.....	28
Ilustración 9: Android 3.0/3.1/3.2 Honeycumb.....	28
Ilustración 10: Android 4.0 Ice Cream Sandwich.....	28
Ilustración 11: Android 4.1/4.2/4.3 Jelly Bean.....	29
Ilustración 12: Arquitectura Android.....	30
Ilustración 13: Nivel de malware en dispositivos móviles.....	31
Ilustración 14: Malware por S.O. (2012).....	32
Ilustración 15: Diagrama de casos de uso.....	37
Ilustración 16: TC1-AnserverBot. Comportamiento ordinario: 1.....	78
Ilustración 17: TC1-AnserverBot. Comportamiento ordinario: 2.....	78
Ilustración 18: TC1-AnserverBot. Resultado final: 1.....	80
Ilustración 19: TC1-AnserverBot. Resultado final: 2.....	81
Ilustración 20: TC2-DroidKungFu. Comportamiento ordinario: 1.....	82
Ilustración 21: TC2-DroidKungFu. Comportamiento ordinario: 2.....	83
Ilustración 22: TC2-DroidKungFu. Resultado final: 1.....	85
Ilustración 23: TC2-DroidKungFu. Resultado final: 2.....	85
Ilustración 24: TC3-GingerMaster. Comportamiento ordinario: 1.....	87
Ilustración 25: TC3-GingerMaster. Comportamiento ordinario: 2.....	87
Ilustración 26: TC3-GingerMaster. Resultado final: 1.....	89
Ilustración 27: TC3-GingerMaster. Resultado final: 2.....	89
Ilustración 28: TC4-DroidKungFu. Comportamiento ordinario: 1.....	90
Ilustración 29: TC4-DroidKungFu. Comportamiento ordinario: 2.....	91
Ilustración 30: TC4-DroidKungFu. Resultado final: 1.....	93

Ilustración 31: TC4-DroidKungFu. Resultado final: 2.....	93
Ilustración 32: TC5-Pjapps. Comportamiento ordinario: 1.	94
Ilustración 33: TC5-Pjapps. Comportamiento ordinario: 2.	95
Ilustración 34: TC5-Pjapps. Conexiones abiertas iniciales.	96
Ilustración 35: TC5-Pjapps. Resultado final: 1.	98
Ilustración 36: TC5-Pjapps. Resultado final: 2.	98
Ilustración 37: TC5-Pjapps. Conexiones abiertas finales.....	99
Ilustración 38: Arquitectura del sistema.	101
Ilustración 39: Diagrama de flujo: Módulo Detección.....	105
Ilustración 40: Diagrama de clases.....	106
Ilustración 41: Clases: Análisis de ofuscación.	107
Ilustración 42: Clases: Análisis de comportamiento.	110
Ilustración 43: Clases: Detección de malware ofuscado.	111
Ilustración 44: Integrantes del sistema.	116
Ilustración 45: Fórmula Entropía de Shannon.....	118
Ilustración 46: Algoritmo Entropía de Shannon.....	118
Ilustración 47: Instrucción llamada criptográfica.	122
Ilustración 48: Instrucción sustitutiva a llamada criptográfica.....	122
Ilustración 49: Nivel de detección.	130
Ilustración 50: ADRD: Artefactos detectados.	131
Ilustración 51: AnserverBot: Artefactos detectados.....	132
Ilustración 52: DroidKungFu: Artefactos detectados.....	132
Ilustración 53: Geinimi: Artefactos detectados.	133
Ilustración 54: jSMShider: Artefactos detectados.	134
Ilustración 55: Pjapps: Artefactos detectados.....	134
Ilustración 56: Diagrama de Gantt: Planificación inicial.	144
Ilustración 57: Diagrama de Gantt: Desarrollo real del proyecto.....	146
Ilustración 58: Comportamiento ordinario: Grupo 1.....	165
Ilustración 59: Comportamiento ordinario: Grupo 2.....	166
Ilustración 60: G1 Assets: L/E.	167
Ilustración 61: G2 Assets: L/E.	168

Ilustración 62: G1 Assets: Net.....	169
Ilustración 63: G2 Assets: Net.....	170
Ilustración 64: G1 Assets: DNS.	171
Ilustración 65: G2 Assets: DNS.	172
Ilustración 66: G1 Assets: Net cifrado.	173
Ilustración 67: G2 Assets: Net cifrado.	173
Ilustración 68: G1 Assets: Net sin cifrar.	174
Ilustración 69: G2 Assets: Net sin cifrar.	175
Ilustración 70: G1 Cadenas ofuscadas: L/E.....	176
Ilustración 71: G2 Cadenas ofuscadas: L/E.....	177
Ilustración 72: G1 Cadenas ofuscadas: Net.	178
Ilustración 73: G2 Cadenas ofuscadas: Net.	179
Ilustración 74: G1 Cadenas ofuscadas: DNS.....	180
Ilustración 75: G2 Cadenas ofuscadas: DNS.....	181
Ilustración 76: G1 Cadenas ofuscadas: Net cifrado.....	182
Ilustración 77: G2 Cadenas ofuscadas: Net cifrado.....	183
Ilustración 78: G1 Cadenas ofuscadas: Net sin cifrar.....	184
Ilustración 79: G2 Cadenas ofuscadas: Net sin cifrar.....	184
Ilustración 80: G1 Cadenas ofuscadas: Op cripto.	185
Ilustración 81: G2 Cadenas ofuscadas: Op cripto.	186
Ilustración 82: G1 Condicionales: L/E.	187
Ilustración 83: G2 Condicionales: L/E.	188
Ilustración 84: G1 Condicionales: Net.	189
Ilustración 85: G2 Condicionales: Net.	190
Ilustración 86: G1 Condicionales: DNS.	191
Ilustración 87: G2 Condicionales: DNS.	192
Ilustración 88: G1 Condicionales: Net cifrado.	193
Ilustración 89: G2 Condicionales: Net cifrado.	193
Ilustración 90: G1 Condicionales: Net sin cifrar.	194
Ilustración 91: G2 Condicionales: Net sin cifrar.	195
Ilustración 92: G1 Condicionales: Op cripto.	196

Ilustración 93: G2 Condicionales: Op cripto.....	197
Ilustración 94: G1 API criptográfica: L/E.	198
Ilustración 95: G2 API criptográfica: L/E.	198
Ilustración 96: G1 API criptográfica: Net.	199
Ilustración 97: G2 API criptográfica: Net.	200
Ilustración 98: G1 API criptográfica: DNS.	201
Ilustración 99: G2 API criptográfica: DNS.	202
Ilustración 100: G1 API criptográfica: Net cifrado.	202
Ilustración 101: G2 API criptográfica: Net cifrado.	203
Ilustración 102: G1 API criptográfica: Net sin cifrar.	204
Ilustración 103: G2 API criptográfica: Net sin cifrar.	204
Ilustración 104: G1 API criptográfica: Op cripto.....	205
Ilustración 105: G2 API criptográfica: Op cripto.....	206

CAPÍTULO 1

INTRODUCCIÓN

1.1 MOTIVACIÓN

Actualmente, existe una fuerte demanda por parte de la sociedad en el uso de dispositivos móviles, como teléfonos inteligentes y tabletas. Esta circunstancia es debida a que este tipo de aparatos ofrecen las prestaciones propias de un terminal móvil, combinadas con las características y medios que puede ofrecer un ordenador convencional.

Una de las plataformas ligadas al desarrollo de los teléfonos inteligentes que más ha crecido en estos últimos años (1) ha sido Android. Esta plataforma es un sistema operativo basado en Linux, diseñado principalmente para teléfonos inteligentes y tabletas, cuya propiedad pertenece a la empresa Google.

Debido a esta fuerte expansión del sistema Android, paralelamente ha proliferado la aparición de *malware* o software malicioso que pone en riesgo la seguridad de esta plataforma. Siendo destacado el incremento de aparición de software maligno desde el verano del año 2010, incrementándose en un 400% para plataformas Android (2).

Esta situación provoca la necesidad de establecer un análisis exhaustivo de las aplicaciones Android en busca de *malware*. Los atacantes han sido capaces de adaptarse a las nuevas características de estas aplicaciones y han conseguido establecer distintas técnicas para llevar a cabo sus acciones maliciosas. Una de las técnicas más explotadas es la ofuscación del código. Esta técnica se basa en la ocultación de componentes integrados en el código maligno que desempeñan una actividad maliciosa (*payload*) para dificultar la detección del código malintencionado por parte de los sistemas de seguridad. La ofuscación se emplea sobre elementos que utiliza el *malware* para llevar a cabo sus fines maliciosos. Estos elementos pueden ser URLs de servidores remotos de comando y control (C&C), informaciones a enviar a servidores remotos, nombres de constantes, palabras clave, nombres de ficheros o fragmentos de código ejecutable de carácter malicioso (*exploit*) (3).

En la actualidad existen numerosas familias de *malware* que utilizan técnicas de ofuscación para llevar a cabo sus acciones maliciosas. Tres de estas familias que afectan a dispositivos Android son:

- **DroidKungFu**: establece una puerta trasera en el dispositivo infectado, lo que permite al atacante robar información sensible presente en el dispositivo. También conecta al dispositivo con una red de dispositivos infectados de forma remota (*botnet*), permitiendo realizar acciones controladas por el atacante, sin necesidad de la autorización del usuario (4). Esta familia presenta variables ofuscadas en su código fuente que se corresponden con direcciones de servidores C&C, además de *exploits* cifrados para dificultar su detección (1).
- **AnserverBot**: es un tipo de troyano que emplea la ofuscación para esconder *payloads* empleados en las actividades maliciosas. En concreto este tipo de *malware*, esconde tras dos ficheros que aparentan ser bases de datos,

almacenados en la carpeta de recursos *assets*, dos aplicaciones que son instaladas en tiempo de ejecución y se encargan de toda la funcionalidad maliciosa (5).

- **Pjapps**: troyano que afecta a dispositivos Android, filtrando información sensible del usuario estableciendo una puerta trasera. Este tipo de *malware* utiliza técnicas de cifrado para ocultar direcciones de servidores de comando y control. En el código fuente se presentan como cadenas de texto las direcciones cifradas de dichos servidores (1).

Las tareas centradas en analizar una aplicación en búsqueda de artefactos ofuscados en su código fuente y esclarecer su contenido original, para identificar si su funcionalidad está ligada a comportamientos maliciosos, resultan labores arduas y laboriosas. Es un trabajo que si se realiza de una manera manual, es necesario invertir grandes cantidades de tiempo.

Con el desarrollo de un sistema que permitiese la automatización de esta labor, se conseguiría aliviar el esfuerzo y reducir el tiempo que tiene que emplear un analista de seguridad para llevar a cabo la detección de artefactos ofuscados con funcionalidad software maliciosa.

1.2 OBJETIVOS

El objetivo principal del TFG consiste en el desarrollo de un sistema que permita la detección, en aplicaciones Android, de artefactos ofuscados con funcionalidad software maliciosa de manera automática.

Una aplicación Android puede presentar artefactos que hayan sido manipulados intencionadamente para que incluyan software malintencionado, que puedan aprovechar un *payload* malicioso. Este TFG se centra en el desarrollo de una técnica eficiente que permita conocer que se esconde tras los fragmentos ofuscados de las aplicaciones. La ofuscación es un acto deliberado de realizar un cambio no destructivo en el código fuente o código máquina, con el fin de que no sea fácil de entender o leer. Por esta razón, esta técnica es utilizada tanto por los desarrolladores de las aplicaciones, para proteger elementos sensibles y evitar su copia, como por atacantes, los cuales alteran elementos del programa y utilizan la ofuscación para evitar el análisis estático del *malware* introducido.

El sistema a desarrollar estará basado en la técnica de *repackaging* o “reenvasado”, que consiste en el desensamblado y reensamblado de las aplicaciones, para la generación de nuevas variantes de la misma *app*. Esta técnica se combina con la mutación de los elementos ofuscados presentes en la aplicación, por lo que será necesario realizar un estudio previo del código fuente de la aplicación (análisis estático) para identificarlos. Por cada artefacto ofuscado mutado, se realizará un caso de prueba distinto,

reensamblando la aplicación con la modificación oportuna. Cada aplicación generada para cada caso de prueba será sometida a un análisis dinámico, para estudiar su comportamiento y las variaciones producidas respecto a la ejecución de la aplicación sin modificar (6).

Por lo tanto los objetivos que se pretenden alcanzar mediante el desarrollo de este Trabajo de Fin de Grado son los siguientes:

- Analizar las principales técnicas de ofuscación empleadas por *malware* en Android.
- Desarrollar un sistema automático basado en casos de prueba capaz de detectar los artefactos ofuscados presentes en una aplicación Android.
- Determinar de manera automatizada si el artefacto ofuscado analizado se trata de una pieza utilizada por el código malicioso para llevar a cabo sus fines malintencionados.
- Utilizar como base la técnica *repackaging*.

1.3 ESTRUCTURA DEL DOCUMENTO

En esta sección se hace una breve presentación del contenido de cada uno de los siete capítulos y dos anexos en los que está organizado el presente documento:

Capítulo 1. Introducción: en este capítulo se muestra la motivación y los objetivos que persigue el desarrollo del Trabajo Fin de Grado realizado.

Capítulo 2. Análisis: en este capítulo se detalla el Estado del arte, en el que se presentan las principales características del sistema Android y las distintas alternativas para analizar estática y dinámicamente aplicaciones. También se presenta un estudio previo realizado para analizar la estructura y comportamiento de distintos tipos de *malware* ofuscado presentes en aplicaciones Android. De los resultados de este estudio se concluyen las bases con las que debe desarrollarse el sistema. También se presenta la especificación de los casos de uso, los requisitos de usuario y los requisitos de software que caracterizan al sistema.

Capítulo 3. Diseño: en este capítulo se detalla la arquitectura del sistema y los componentes que lo forman, a través de un diagrama de componentes. También se explica el diagrama de clases que muestra la estructura del sistema.

Capítulo 4. Implementación: en este capítulo se detalla los aspectos más trascendentes llevados a cabo durante la codificación del sistema.

Capítulo 5. Evaluación: en este capítulo se presenta los resultados de detección de *malware* ofuscado obtenidos tras analizar un conjunto de aplicaciones Android utilizando el sistema desarrollado.

Capítulo 6. Conclusiones: en este capítulo se expone las reflexiones y resoluciones obtenidas tras la realización de este Trabajo Fin de Grado.

Capítulo 7. Gestión del proyecto: en este capítulo se muestra la planificación inicial del proyecto junto con el seguimiento del mismo. Adicionalmente, se detalla el presupuesto del proyecto y el coste real del mismo, junto con las desviaciones producidas durante el desarrollo del mismo.

Anexo I. Bibliografía: en este anexo se expone la bibliografía en la elaboración de este documento.

Anexo II. Evaluación: Resultados de comportamiento: en este anexo se presenta los resultados obtenidos en la Evaluación enfocados al comportamiento de las aplicaciones.

1.4 ACRÓNIMOS

En esta sección se indica todos los acrónimos utilizados en el presente documento, organizados por orden alfabético, y su significado:

AOSP: *Android Open Source Project.*

API: *Application Programming Interface.*

APK: *Application Package File.*

AVD: *Android Virtual Device.*

C&C: *Command and Control.*

CU: Caso de Uso.

DNS: *Domain Name System.*

GPS: *Global Positioning System.*

HTTP: *Hypertext Transfer Protocol.*

JPG: *Joint Photographic Experts Group.*

NFC: *Near Field Communication.*

OHA: *Open Handset Alliance.*

OpenGL/SL: *OpenGL Shading Language.*

PNG: *Portable Network Graphics.*

RS: Requisito de Software.

RSF: Requisito Software Funcional.

RSNF: Requisito Software No Funcional.

RU: Requisito de Usuario.

SD: *Secure Digital.*

SDK: *Software Development Kit.*

SGL: *Scene Graph Library.*

SIP: *Session Initiation Protocol.*

SMS: *Short Message Service.*

SSL: *Secure Sockets Layer.*

TCP: *Transmission Control Protocol.*

TFG: Trabajo Fin de Grado.

UI: *User Interface.*

UML: *Unified Modeling Language.*

URL: *Uniform Resource Locator.*

VoIP: *Voice over IP.*

VPN: *Virtual Private Network.*

Wi-Fi: *Wireless Fidelity.*

XML: *Extensible Markup Language.*

XMPP: *Extensible Messaging and Presence Protocol.*

1.5 GLOSARIO DE TÉRMINOS

Esta sección presenta los términos principales utilizados en el presente documento:

Análisis dinámico: estudio del comportamiento de un programa en ejecución.

Análisis estático: análisis de software que se realiza sin ejecutar el programa, estudiando el código fuente.

Android: sistema operativo basado en Linux, diseñado para ser utilizado en dispositivos móviles.

Assets: directorio utilizado por las aplicaciones Android para almacenar recursos.

Botnet: red formada por distintos sistemas de información conectados entre sí que son manipulados ilegítimamente por un atacante para llevar a cabo acciones maliciosas.

Caso de prueba/Test case: cada una de las ejecuciones de la aplicación a analizar realizadas bajo unas modificaciones determinadas por una serie de condiciones.

Exploit: fragmento de software o comandos que aprovechan una vulnerabilidad de seguridad de un sistema de información para conseguir un comportamiento no deseado del mismo.

Malware: tipo de software malicioso cuyo objetivo es infiltrarse o dañar un sistema de información.

Ofuscación: técnica utilizada para encubrir el significado de una información haciéndola más confusa y complicada de interpretar.

Payload: parte de un programa maligno que realiza una acción maliciosa.

Repackaging: técnica basado en la descompilación de un programa para modificar su código fuente, para finalmente volver a compilarla con dicha modificación.

Script: programa escrito en lenguaje interpretado y almacenado en un fichero de texto plano.

CAPÍTULO 2

ANÁLISIS

2.1 DEFINICIÓN DEL SISTEMA

2.1.1 ALCANCE DEL SISTEMA

El principal problema a solventar con el desarrollo de este proyecto consiste en la detección de comportamiento malicioso que pueda hallarse tras artefactos ofuscados en una aplicación Android.

Con este sistema se proporciona una herramienta automática de análisis de este tipo de código malintencionado, basándose en el uso de casos de prueba. Generalmente, un caso de prueba es un conjunto de condiciones o variables dadas para un objetivo en particular. En este caso, un caso de prueba se refiere a cada una de las ejecuciones de la aplicación a analizar realizadas bajo unas modificaciones determinadas por una serie de condiciones.

Con los resultados obtenidos tras la batería de casos de prueba, se detecta automáticamente si la aplicación analizada presenta *malware* ofuscado o no en su código.

2.1.2 ESTÁNDARES Y NORMAS

Los distintos diagramas presentes en este documento, utilizados para la definición del sistema, serán desarrollados mediante el lenguaje de modelado de sistemas software UML (7).

Los elementos de UML empleados han sido los siguientes:

- Diagrama de casos de uso.
- Diagrama de clases.
- Diagrama de componentes
- Diagrama de flujo.

2.2 ESTADO DEL ARTE

2.2.1 DIFUSIÓN DEL SISTEMA OPERATIVO ANDROID

Android es un sistema operativo basado en Linux, diseñado para ser utilizado en dispositivos móviles, como teléfonos inteligentes o tabletas.

En la Ilustración 1 (8) se observa la cuota de mercado mundial de teléfonos inteligentes, desde principios del año 2011 hasta finales del año 2012. Los teléfonos inteligentes con el sistema Android presentan un crecimiento casi constante en torno a cada uno de los cuatrimestres.

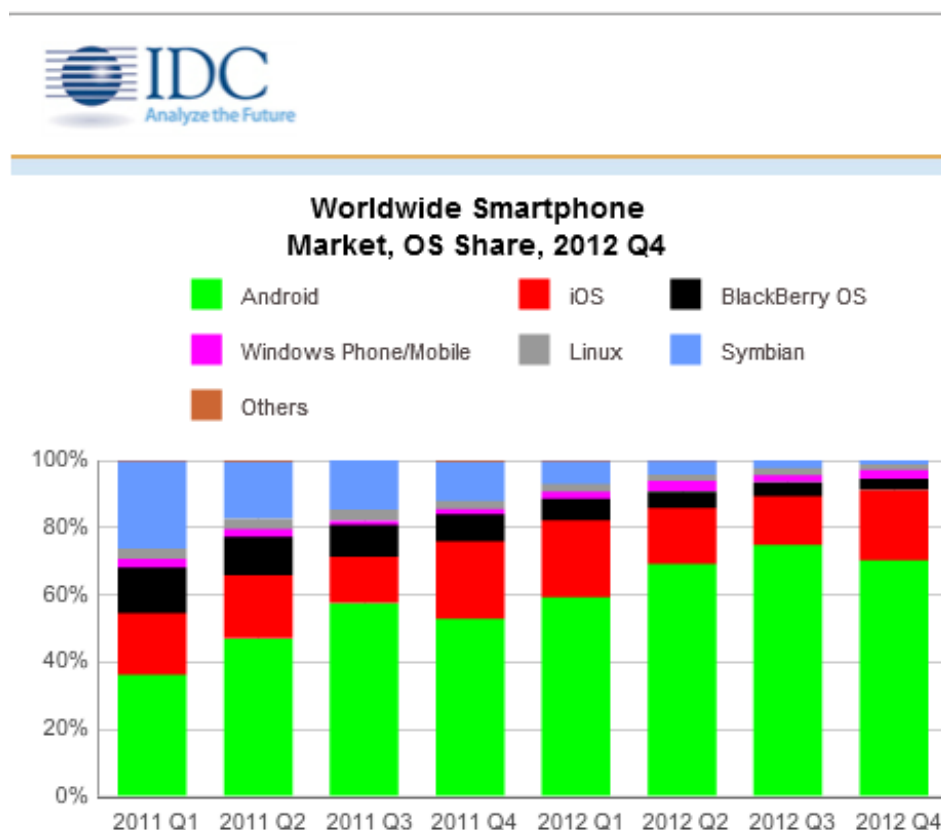


Ilustración 1: Cuota de mercado mundial de smartphones: 2011/2012

El sistema que ha presentado un mayor crecimiento ha sido el sistema Android respecto al resto de sistemas operativos en teléfonos inteligentes.

En las Ilustraciones 2 (9) y 3 (10) se presenta el volumen de ventas y el porcentaje de mercado de los cinco sistemas operativos más utilizados, tanto en teléfonos inteligentes como en tabletas, en el primer cuatrimestre del año 2013.

Top Five Smartphone Operating Systems, Shipments, and Market Share, 1Q 2013 (Units in Millions)

Operating System	1Q13 Shipment Volume	1Q13 Market Share	1Q12 Shipment Volume	1Q12 Market Share	Year over Year Change
Android	162.1	75.0%	90.3	59.1%	79.5%
iOS	37.4	17.3%	35.1	23.0%	6.6%
Windows Phone	7.0	3.2%	3.0	2.0%	133.3%
BlackBerry OS	6.3	2.9%	9.7	6.4%	-35.1%
Linux	2.1	1.0%	3.6	2.4%	-41.7%
Symbian	1.2	0.6%	10.4	6.8%	-88.5%
Others	0.1	0.0%	0.6	0.4%	-83.3%
Total	216.2	100.0%	152.7	100.0%	41.6%

Ilustración 2: Top 5 SSOO: Smartphones (1Q 2013)**Top Tablet Operating Systems, Shipments, and Market Share, 2013 Q1 (Shipments in Millions)**

Vendor	1Q13 Unit Shipments	1Q13 Market Share	1Q12 Unit Shipments	1Q12 Market Share	Year-over-Year Growth
Android	27.8	56.5%	8.0	39.4%	247.5%
iOS	19.5	39.6%	11.8	58.1%	65.3%
Windows	1.6	3.3%	0.2	1.0%	700.0%
Windows RT	0.2	0.4%	0.0	N/A	N/A
Others	0.1	0.2%	0.2	1.0%	-50.0%
Total	49.2	100.0%	20.3	100.0%	142.4%

Ilustración 3: Top 5 SSOO: Tablets (1Q 2013)

En los dos tipos de dispositivos móviles, el líder indiscutible es el sistema operativo Android presentando un 75% de la cuota de mercado en teléfonos inteligentes y un 56,5% en tabletas.

Estos datos afirman la fuerte expansión de este sistema operativo en el mercado, siendo el más utilizado por los consumidores de este tipo de dispositivos.

Esta situación ha provocado que este sistema sea uno de los objetivos principales de los atacantes.

2.2.2 ANDROID: ARQUITECTURA Y CARACTERÍSTICAS

El sistema Android nació como una variante Linux destinada a dispositivos móviles. Esta plataforma fue creada por la compañía Android Inc., la cual fue comprada por Google en el año 2005. La plataforma fue liberada bajo el nombre de *Android Open Source Project* (AOSP) en el año 2007. Un grupo de 78 compañías distintas formaron la *Open Handset Alliance* (OHA), cuyo trabajo se centró en el desarrollo y distribución del sistema Android (11).

Las aplicaciones Android se ejecutan en máquinas virtuales, independientes unas de otras. La máquina virtual utilizada por Android se denomina Dalvik, la cual es capaz de ejecutar su propio código de bytes. La mayoría de las aplicaciones Android están escritas en Java y son ejecutadas en la máquina virtual Dalvik.

Las versiones de la plataforma Android liberadas hasta el momento han sido las siguientes:

- **Android 1.0 Android Pie:** liberada en septiembre de 2008. Presenta una interfaz de usuario básica y limitaciones en el uso de la cámara, Bluetooth y Flash (12).
- **Android 1.1 Banana Bread:** lanzada en febrero del año 2009. Esta versión solucionó algunos problemas de la versión Android Pie y agregó nuevas funcionalidades, como la posibilidad de guardar archivos adjuntos en los mensajes (12).
- **Android 1.5 Cupcake:** versión liberada en el mes de abril del 2009. Todos los elementos básicos de la interfaz de usuario fueron actualizados, también se incorporó aplicaciones basadas en acelerómetro, grabación de video y reproducción, así como compatibilidades con nuevas versiones de Bluetooth (11).



Ilustración 4: Android 1.5 Cupcake.

- **Android 1.6 Donut:** lanzada en septiembre de 2009. Se añadió soporte a pantallas de mayor resolución y se incorporó un motor de texto a voz. También se incorporó soporte con VPN & 802.1x (11).



Ilustración 5: Android 1.6 Donut.

- **Android 2.0/2.1 Éclair:** versión 2.0 liberada en octubre de 2009 y versión 2.1 en enero de 2010. Se añadieron mejoras en la interfaz de usuario, compatibilidad con Bluetooth 2.1 y soporte para Microsoft Exchange (11).



Ilustración 6: Android 2.0/2.1 Éclair.

- **Android 2.2 Froyo:** lanzado en mayo de 2010. Presenta optimizaciones de rendimiento, soporte para Adobe Flash, soporte mejorado para Microsoft Exchange y soporte para OpenGL ES 2.0 (11).



Ilustración 7: Android 2.2 Froyo.

- **Android 2.3.x Gingerbread:** liberada en diciembre de 2010. Está basada en el núcleo Linux 2.6.35. En cuanto a novedades, se actualizó el diseño de la interfaz de usuario mejorando la velocidad y la legibilidad. También se incorporó soporte para NFC y mejoras para la funcionalidad de copiar/pegar, así como soporte para VoIP y SIP (13).



Ilustración 8: Android 2.3.x Gingerbread.

- **Android 3.0/3.1/3.2 Honeycomb:** lanzada en febrero de 2011. Basada en el kernel Linux 2.6.36. Fue una actualización exclusiva para tabletas Android, optimizando la interfaz a las características de este tipo de dispositivos móviles (14).



Ilustración 9: Android 3.0/3.1/3.2 Honeycomb.

- **Android 4.0 Ice Cream Sandwich:** lanzado públicamente en octubre de 2011. Basada en el kernel Linux 3.0.1. Algunas de sus principales novedades son: mejoras de estabilidad, aumento de rapidez en la interfaz de usuario y soporte con Wi-Fi Direct (15).



Ilustración 10: Android 4.0 Ice Cream Sandwich.

- **Android 4.1/4.2/4.3 Jelly Bean:** la versión 4.1 fue liberada en junio de 2012, la versión 4.2 en noviembre de 2012 y la versión 4.3 en julio de 2013. Sus mejoras se centran principalmente en optimizar el rendimiento y la funcionalidad de la interfaz de usuario. También se incorporó soporte para nuevas versiones de Bluetooth y OpenGL (16).



Ilustración 11: Android 4.1/4.2/4.3 Jelly Bean.

En cuanto a la arquitectura utilizada por la plataforma Android, cabe destacar su organización basado en capas. Las cinco capas que forman el sistema Android, representadas en la Ilustración 12, son:

- **Aplicaciones:** este nivel incluye todas las aplicaciones que el usuario utiliza. Las aplicaciones utilizan servicios, APIs y librerías de los niveles inferiores.
- **Framework de aplicaciones:** en este nivel se encuentran las herramientas de desarrollo de cualquier aplicación. La mayoría de estas herramientas son bibliotecas Java que acceden a recursos desde la máquina Dalvik. Las más importantes son las siguientes:
 - Activity Manager: se centra en la gestión del ciclo de vida de las actividades y de la pila de actividades.
 - Windows Manager: gestiona las ventanas de las aplicaciones.
 - Telephone Manager: controla todas las funcionalidades propias del teléfono, como llamadas o mensajes.
 - Location Manager: relacionado con la gestión de servicios de localización y posicionamiento.
 - Notification Manager: gestión de notificaciones para avisar al usuario de distintos sucesos, como llamadas entrantes o mensajes recibidos.
 - XMPP Service: colección de APIs para utilizar este protocolo de intercambio basado en el lenguaje XML.
- **Librerías:** esta capa está formada por librerías utilizadas por el sistema Android, proporcionando la mayor parte de sus capacidades. Las librerías más destacables son las siguientes:
 - Librería libc: formada por todas las cabeceras y funciones según el estándar de lenguaje C.

- OpenGL/SL y SGL: representan las librerías gráficas del sistema Android.
- Bibliotecas multimedia: proporcionan los códec necesarios para reproducir el contenido multimedia del dispositivo.
- SQLite: motor de bases de datos relacionales utilizado por las aplicaciones.
- FreeType: permite trabajar con distintos tipos de fuentes tipográficas.
- Librería SSL: permite gestionar comunicaciones seguras utilizando dicho protocolo.
- **Entorno de ejecución:** se encuentra al mismo nivel que las librerías de Android. Formada por las Core Libraries, que incluye las bibliotecas habituales de Java y específicas de Android, y la máquina virtual Dalvik.
- **Kernel de Linux:** capa utilizada para gestionar los componentes hardware del dispositivo móvil a través de llamadas al sistema. Se encarga de gestionar recursos como la cámara, la memoria o el audio, entre otros.

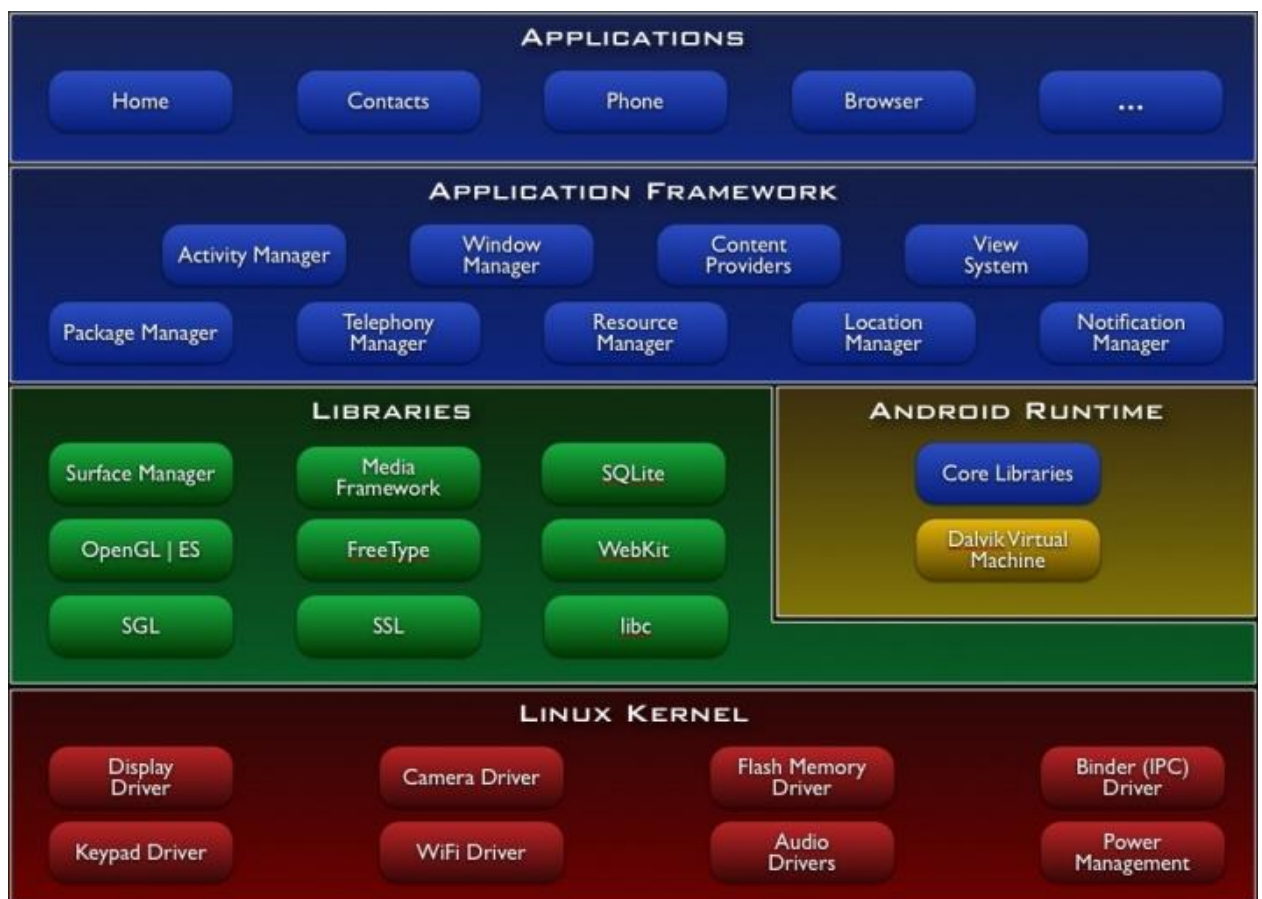


Ilustración 12: Arquitectura Android.

2.2.3 MALWARE EN ANDROID

El modelo de seguridad en sistemas Android está basado en el aislamiento de procesos y en un mecanismo de permisos.

Cada aplicación se ejecuta en un proceso único e independiente del resto, creando una capa de seguridad entre el sistema y las aplicaciones a nivel de proceso. Cada aplicación se ejecuta de manera independiente a las otras utilizando una Dalvik Virtual Machine, la cual no posee acceso a los ficheros ni recursos de otras aplicaciones.

También se utiliza un mecanismo de permisos que otorga un nivel adicional de seguridad al sistema. A cada aplicación se le puede conceder una serie de permisos para el acceso a características protegidas del sistema, los cuales son otorgados por el usuario final en el momento de la instalación (17). Asimismo, cada aplicación Android debe estar firmada con una clave privada que pertenece al desarrollador para poder ser distribuida en mercados oficiales, para evitar ataques contra la integridad (18).

En la Ilustración 13 se presenta el incremento de *malware* en dispositivos móviles, independientes del sistema operativo, desde el año 2004 hasta el año 2012 (17):

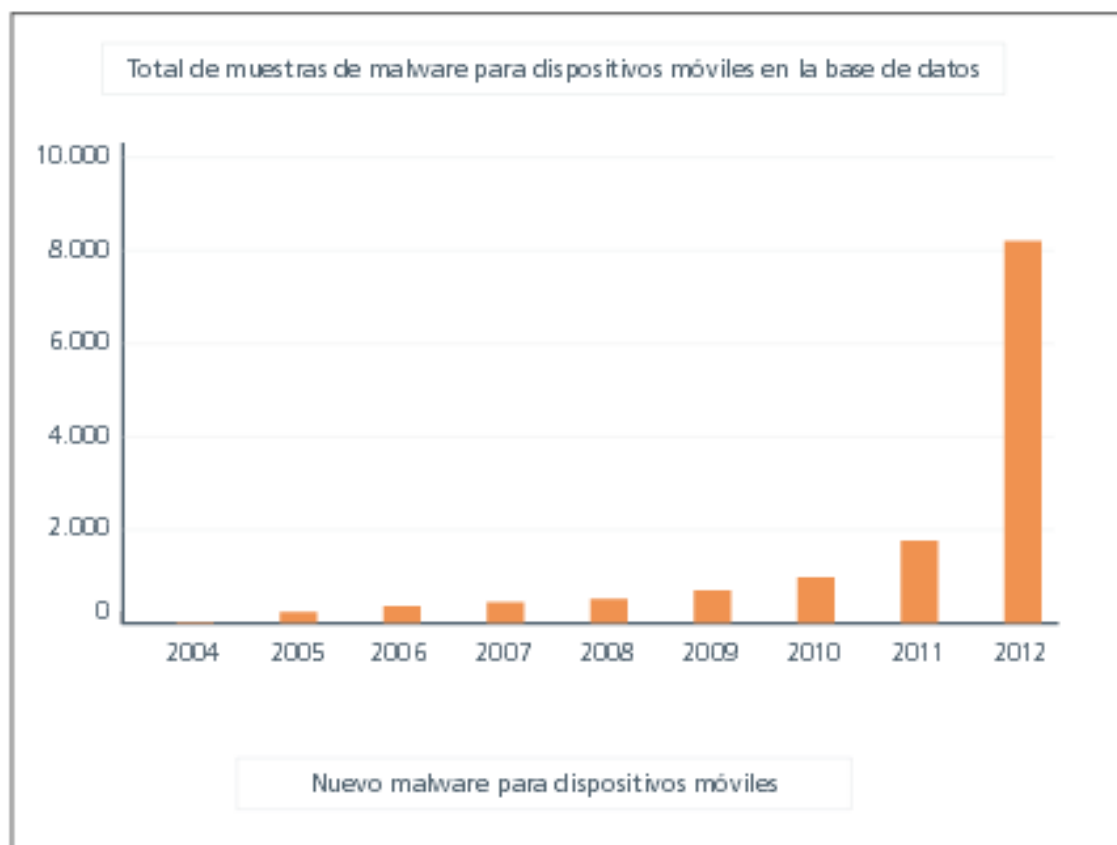


Ilustración 13: Nivel de malware en dispositivos móviles.

La aparición de *malware*, a niveles generales, ha sufrido un significativo crecimiento con el paso de los años, siendo notorio el incremento producido durante el año 2012.

La fuerte acogida de sistemas Android por parte de los usuarios de dispositivos móviles, ha sido uno de los puntos clave para que los atacantes se hayan centrado en investigar y explotar las vulnerabilidades del modelo de seguridad utilizado por Android. Cabe destacar el incremento de *malware* en más de un 400%, desde el verano del año 2010, en dispositivos móviles que utilizan el sistema Android (2).

En la Ilustración 14 (17) se muestra el porcentaje de *malware* que afecta a cada uno de los distintos sistemas operativos utilizados en dispositivos móviles, en el año 2012:

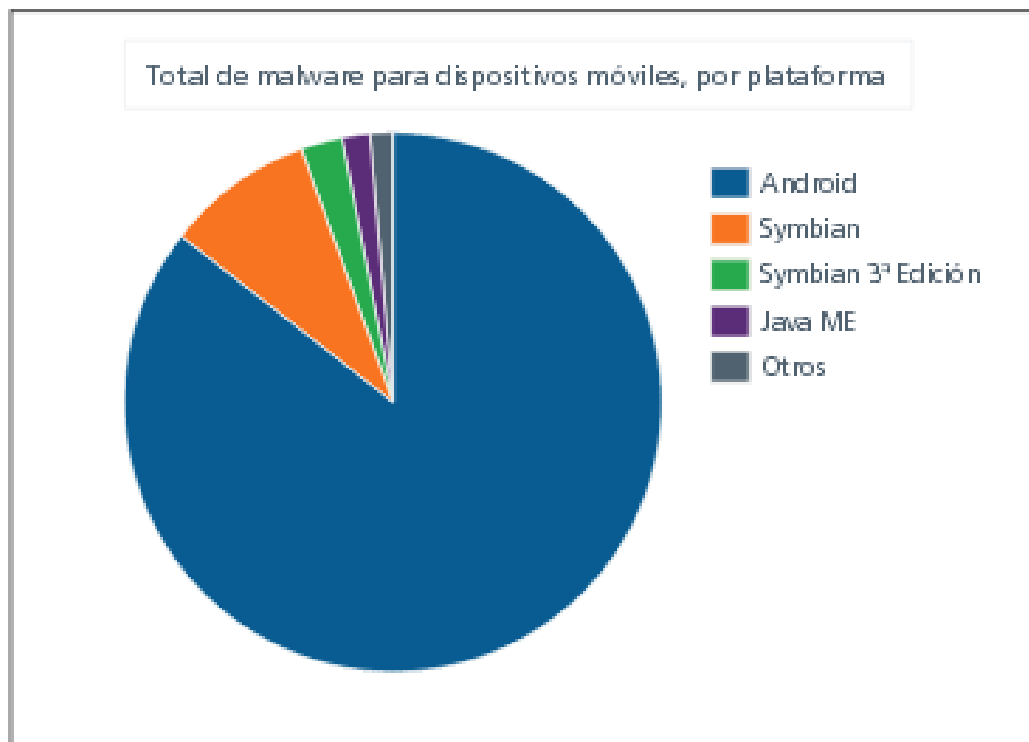


Ilustración 14: Malware por S.O. (2012)

La plataforma Android es la que mayor porcentaje de *malware* presenta respecto a otros sistemas operativos.

El código malicioso que afecta a sistemas Android aprovecha vulnerabilidades presentes en el mecanismo de permisos, para poder escalar privilegios y obtener acceso a funcionalidades protegidas del sistema, como pueden ser el envío de SMS o llamadas telefónicas, acceso a la localización GPS o a la tarjeta micro SD, entre otras (18).

Una de las principales técnicas utilizadas en la elaboración de código malicioso se trata de la ofuscación. Esta técnica consiste en el acto deliberado de realizar un cambio no destructivo en un elemento del sistema con el fin de dificultar su entendimiento o legibilidad (19).

Existen distintos métodos para llevar a cabo este tipo de técnica, los hay que emplean la criptografía, cifrando y descifrando cadenas de texto utilizando una clave (19), y otros emplean la esteganografía, con el fin de ocultar código malicioso en distintos tipos de recursos, como las imágenes (20).

2.2.4 INGENIERÍA INVERSA APLICADA SOBRE MALWARE EN ANDROID

El análisis estático es un tipo de análisis de software que se realiza sin la ejecución del programa, centrándose en estudiar el código fuente o/y código objeto.

La ingeniería inversa consiste en la capacidad de generar el código fuente a partir del archivo ejecutable. Es utilizado para analizar el funcionamiento de un programa, para evadir mecanismos de seguridad, modificar el programa original, etc.

Las aplicaciones Android son distribuidas en formato pre-compilado binario Dalvik, por lo que no es posible acceder directamente al código fuente. Pero existen herramientas, basadas en ingeniería inversa, que actúan como desensambladores que convierten el código binario Dalvik, que se encuentra en formato *Dex*, a un formato legible para una persona (18).

Una de las principales técnicas basada en ingeniería inversa es el *repackaging* o “re-empaquetamiento”. Consiste en desensamblar una aplicación para modificar su código fuente o agregar nuevo, y volver a re-ensamblar la aplicación alterada. Esta es una de las técnicas más empleadas para agregar código malicioso en aplicaciones legítimas Android, que son distribuidas a través de mercados de aplicaciones alternativos (18).

Actualmente, existen distintas herramientas que permiten realizar un análisis estático de las aplicaciones que se basan en esta técnica de ingeniería inversa:

- **ApkTool:** herramienta de ingeniería inversa capaz de analizar los códigos binarios de aplicaciones Android (21). Sus principales características son:
 - Desensamblado de aplicaciones prácticamente en su forma original, y re-empaquetamiento después de modificar el código desensamblado.
 - Depuración del código *smali*.
 - Inclusión de soporte y características para plataformas de clientes y localización.
 - Automatización de tareas repetitivas, ayudando al usuario a trabajar con las aplicaciones de una manera más cómoda.
- **Dex2Jar:** herramienta desarrollada para transformar los ficheros *dex*, utilizados por la Dalvik Virtual Machine, en ficheros en formato *class* (22). De esta manera se obtiene el código fuente de una aplicación Android en código Java. Esta herramienta es utilizada para poder realizar análisis estáticos de aplicaciones.

Esta herramienta también ofrece otras funcionalidades, como puede ser la firma de aplicaciones Android, descifrado de cadenas de texto presentes en la aplicación, etc

- **JD-GUI:** herramienta basada en ingeniería inversa utilizada para descompilar código Java (23). Sólo puede ser utilizado a través de la interfaz gráfica ofrecida.
- **Smali/baksmali:** es un ensamblador/desensamblador para el formato *dex* utilizado por la máquina virtual Dalvik (24). Está basado en la sintaxis *Jasmin's/dedexer's*.
- **AntiLVL:** herramienta destinada para subvertir los métodos de protección de licencia estándar y deshabilitar métodos de anti-cracking presentes en aplicaciones. Permite descompilar aplicaciones Android, definir búsquedas utilizando expresiones regulares, modificar artefactos presentes en el código fuente, y re-compilar la aplicación con las modificaciones realizadas (25).
- **Radare2:** compuesta por un editor hexadecimal como piedra angular, que permite ensamblar/desensamblar, analizar código, scripting y generar gráficas de los análisis de código y datos. Una de sus características es su fácil integración con Unix (26).

2.2.5 ANÁLISIS DINÁMICO DE APLICACIONES ANDROID.

El análisis dinámico es un tipo de análisis de software cuyo objetivo es estudiar el comportamiento de un cierto programa en ejecución.

Para aplicaciones Android, existen dos herramientas principales para realizar análisis dinámicos de las mismas. Estas herramientas son DroidBox y Droidscope:

- **DroidBox:** herramienta desarrollada para realizar un análisis dinámico de aplicaciones Android (27). DroidBox recoge información sobre distintos comportamientos y acontecimientos ocurridos durante la ejecución de la aplicación Android. La información recogida por DroidBox se centra en:
 - Operaciones de lectura y escritura en ficheros.
 - Actividades criptográficas.
 - Conexiones de red abiertas.
 - Tráfico de red saliente.
 - Información filtrada.
 - Envíos de SMS.
 - Llamadas telefónicas realizadas.

DroidBox se apoya en el SDK de Android para realizar los análisis dinámicos.

- **DroidScope:** se trata de una plataforma de análisis dinámico no basado en el emulador de Android (28).

DroidScope es capaz de reconstruir semánticas a nivel del sistema operativo y al nivel de Java simultáneamente. La API ofrecida consta de tres niveles, cada uno de los cuales reflejan cada uno de los niveles de un dispositivo Android: hardware, sistema operativo y Dalvik Virtual Machine. También contiene distintas herramientas de análisis para recolectar trazas de instrucciones Dalvik, llamadas API a nivel de perfil, y de fuga de información tanto a nivel de Java como al nivel de componentes nativos (29).

A su vez, existen otros tipos de herramientas que complementan el propósito del análisis dinámico con distintas funcionalidades:

- **Tcpdump:** herramienta utilizada para analizar tráfico de red basado en la librería *libpcap*. Permite capturar los paquetes entrantes y salientes de un determinado host. Esta aplicación permite analizar la actividad de red de un programa en ejecución, por lo que puede ser empleada para desarrollar análisis dinámicos centrados en las operaciones de red (30).

Las siguientes herramientas permiten realizar un testeo automático de la interfaz de usuario de las aplicaciones Android. Dependiendo del tipo de interacción que se tenga con la interfaz de la aplicación, se generarán unos comportamientos u otros, por lo que este tipo de instrumentos son fundamentales en un análisis dinámico automático:

- **monkeyrunner:** herramienta proporcionada por el kit de desarrollo de Android. Permite instalar y ejecutar aplicaciones Android (31). Es capaz de interactuar con la interfaz de usuario de las aplicaciones. Sus principales características son las siguientes:
 - Capacidad de lanzar actividades, pulsar botones y rellenar entradas de texto de manera automática.
 - Limitaciones para determinar el botón o entrada de texto de la interfaz que se debe pulsar, ya que solo acepta coordenadas espaciales para la identificación de estos artefactos.
 - Implementado en Python.

Las limitaciones de esta herramienta, principalmente a la hora de poder acceder a los elementos de la interfaz, ya que únicamente se pueden manipular si se conoce las coordenadas espaciales de las mismas, la convierten en una alternativa poco viable para ser integrada en el sistema.

- **AndroidViewClient:** es una herramienta basada en monkeyrunner que extiende su funcionalidad otorgándole las siguientes características (32):
 - Compatibilidad completa con monkeyrunner y su funcionalidad.
 - Identificación de elementos de la interfaz de la actividad Android por su identificador.

- Permite obtener el valor de la mayoría de las propiedades de los elementos de la interfaz.
- Permite enviar eventos de pulsado a los distintos elementos de la interfaz.
- Incorpora herramientas para la detección automática de los elementos de la interfaz presente en pantalla.
- Implementado en Python.

Esta herramienta solventa las limitaciones presentes en *monkeyrunner*, pudiendo interactuar con los artefactos de la interfaz utilizando su identificador. Estos identificadores se pueden obtener a través de los ficheros XML que almacenan las interfaces de las actividades obtenidos al decompilar la aplicación o a través de la aplicación *culebra* incorporada en dicha herramienta, la cual retorna un esquema con todos los elementos de la interfaz y sus propiedades.

Pero su único y principal inconveniente es que no es compatible con la herramienta *Droidbox*.

- **Robotium:** framework de código abierto para la realización de pruebas de caja negra de la interfaz de aplicaciones Android de manera automática. Sus principales características son las siguientes (33):
 - Compatibilidad con las clases de la API de Android
 - Su API presenta métodos capaces de recuperar los identificadores de todo tipo de elementos de la UI de manera automática.
 - Interactividad con todo tipo de artefactos de la interfaz.
 - Implementado en Java.

Robotium ofrece prestaciones similares a *AndroidViewClient*, superando las limitaciones de *monkeyrunner*, y es completamente compatible con la herramienta *Droidbox*.

2.3 CASOS DE USO

2.3.1 DIAGRAMA DE CASOS DE USO

La Ilustración 15 muestra el diagrama de casos de uso del sistema.

El único actor que interactúa con el sistema es *Experto en Seguridad*, que se trata de la persona que utiliza el sistema para obtener un análisis de *malware* ofuscado de una determinada aplicación Android.

El diagrama presenta los casos de uso presentes en el sistema:

- **Analizar aplicación:** consiste en la secuencia de operaciones que se realizan para analizar la aplicación indicada por el *Experto en Seguridad*.
- **Elección del tiempo de prueba del test case:** al actor se le ofrece la opcionalidad de seleccionar el tiempo de prueba empleado para la realización de los casos de prueba llevados a cabo durante el análisis de la aplicación.
- **Visualizar resultados:** consiste en el acceso e interpretación de los datos almacenados en los ficheros de resultados.

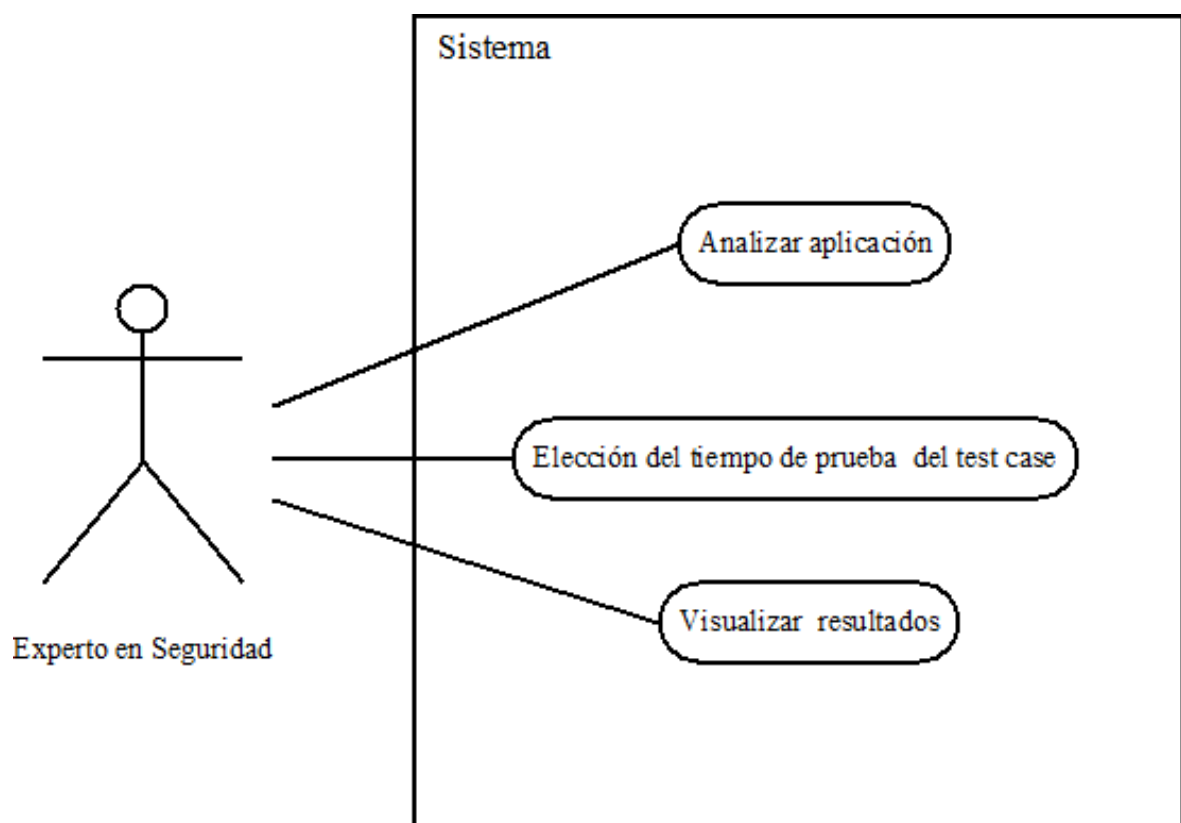


Ilustración 15: Diagrama de casos de uso.

2.3.2 ESPECIFICACIÓN DE ACTORES DEL SISTEMA

En la Tabla 1 se describe el único actor que interactúa con el sistema:

Nombre		Experto en Seguridad
ID	A1.	
Descripción	Rol que toma la persona que utiliza el sistema. El <i>Experto en Seguridad</i> debe conocer como ejecutar un programa a través de línea de comandos, además de conocer los parámetros necesarios. Su objetivo es obtener un análisis sobre la presencia de <i>malware</i> ofuscado de una determinada aplicación.	

Tabla 1: Actores del Sistema: A1.

2.3.3 ESPECIFICACIÓN DE CASOS DE USO DEL SISTEMA

2.3.3.1 ANALIZAR APLICACIÓN (TIEMPO DE PRUEBA POR DEFECTO)

CU-01		Analizar aplicación
Descripción	El Experto en Seguridad pretende analizar una aplicación Android con el fin de obtener los resultados relacionados con la presencia de <i>payloads</i> maliciosos.	
Actor	Experto en Seguridad	
Precondiciones	<ul style="list-style-type: none"> - El sistema debe tener instalado todos los componentes auxiliares utilizados. - La aplicación Android debe estar en formato <i>apk</i>. 	
Flujo de acciones	Paso	Acción
	1	El actor abre la ventana de comandos del sistema.
	2	El sistema muestra la ventana de comandos.
	3	El actor se establece en el directorio en el que se encuentran el programa y el fichero <i>apk</i> .
	4	La ventana de comandos se posiciona en el directorio indicado por el usuario.
	5	El actor introduce el comando para ejecutar el programa de casos de prueba, sin indicar el tiempo de ejecución del

		análisis dinámico: “java -jar TFG.jar <i>nombre_aplicacion.apk</i> ”
	6	El sistema realiza las operaciones pertinentes para analizar la aplicación.
	7	El sistema almacena los resultados en el fichero.
Postcondiciones	- El sistema debe haber generado el fichero donde se plasman los resultados.	
Excepciones	Paso	Acción
	5	El archivo indicado para ser analizado no es una aplicación Android. A continuación este caso de uso termina.

Tabla 2: Caso de uso: CU-01.

2.3.3.2 ELECCIÓN DEL TIEMPO DE PRUEBA DEL TEST CASE

CU-02	Elección del tiempo de prueba del test case	
Descripción	El actor decide el tiempo empleado para la realización de cada caso de prueba.	
Actor	Experto en Seguridad	
Precondiciones	<ul style="list-style-type: none"> - El sistema debe tener instalado todos los componentes auxiliares utilizados. - La aplicación Android debe estar en formato <i>apk</i>. 	
Flujo de acciones	Paso	Acción
	1	El actor abre la ventana de comandos del sistema.
	2	El sistema muestra la ventana de comandos.
	3	El actor se establece en el directorio en el que se encuentran el programa y el fichero <i>apk</i> .
	4	La ventana de comandos se posiciona en el directorio indicado por el usuario.

	5	El actor introduce el comando para ejecutar el programa de casos de prueba indicando el tiempo de ejecución de cada prueba (X): “java -jar TFG.jar <i>nombre_aplicacion.apk</i> X”
	6	El sistema realiza las operaciones pertinentes para analizar la aplicación.
	7	El sistema almacena los resultados en el fichero.
Postcondiciones	-	El sistema debe haber generado el fichero donde se plasman los resultados.
Excepciones	Paso	Acción
	5	El archivo indicado para ser analizado no es una aplicación Android. A continuación este caso de uso termina.
	5	El tiempo de ejecución seleccionado es un número menor que cero. A continuación el caso de uso termina.

Tabla 3: Caso de uso: CU-02.

2.3.3.3 VISUALIZAR RESULTADOS

CU-03	Visualizar resultados	
Descripción	El actor accede al fichero donde se han almacenado los resultados resultantes del análisis.	
Actor	Experto en Seguridad	
Precondiciones	<ul style="list-style-type: none"> - El sistema debe tener instalado todos los componentes auxiliares utilizados. - El sistema debe contar con un programa capaz de visualizar hojas de cálculo. - Se debe de haber ejecutado el sistema a desarrollar con la aplicación a analizar. - El sistema debe haber generado el fichero donde se plasman los resultados 	
Flujo de acciones	Paso	Acción
	1	El actor accede al directorio en donde se encuentra el fichero de resultados deseado.
	2	El sistema muestra dicho directorio.

	3	El actor abre dicho fichero utilizando un programa capaz de visualizar hojas de cálculo.
	4	Se muestra por pantalla la hoja de cálculo generada por el sistema a desarrollar con todos los datos almacenados del análisis.
	5	El actor lee e interpreta los datos recogidos.
Postcondiciones	-	
Excepciones	Paso	Acción
	4	El programa no es capaz de interpretar la hoja de cálculo generada. A continuación el caso de uso termina.

Tabla 4: Caso de uso: CU-03.

2.4 REQUISITOS DEL SISTEMA

2.4.1 ESTRUCTURA DEL REQUISITO

Los requisitos que se presentan en este documento se definen por medio de una ficha en la que se definen sus particularidades. Los atributos que se destacan y sus posibles valores son los siguientes:

- Id: Es el identificador unívoco del requisito.
- Nombre: Nombre identificativo del requisito.
- Descripción: Explicación breve del requisito.
- Estabilidad: Estimación de la probabilidad del cambio del requisito a lo largo del proyecto. Los posibles valores que puede tomar son:
 - Alto: indica que es muy poco probable que el requisito varíe.
 - Medio: indica que es algo probable que el requisito se modifique.
 - Bajo: indica que es muy probable que el requisito cambie.
- Impacto de estabilidad: indica el impacto que puede tener la modificación a posteriori del requisito. Los valores que puede adoptar son:
 - Alto: su variabilidad supone un fuerte impacto a la estabilidad del proyecto.
 - Medio: su variabilidad supone un impacto regular a la estabilidad del proyecto.
 - Bajo: su variabilidad supone un leve impacto a la estabilidad del proyecto.
- Fuente: origen o responsable del requisito.

2.4.2 REQUISITOS DE USUARIO

En esta sección se van a definir los requisitos de usuario definidos del sistema:

RU01	
Nombre: Aplicaciones Android	Fuente: Guillermo Nicolás Suárez de Tangil
Descripción: El sistema debe operar con aplicaciones Android.	
Estabilidad: Alta	Impacto de estabilidad: Alto

Tabla 5: RU01. Aplicaciones Android.

RU02	
Nombre: Análisis de código ofuscado	Fuente: Guillermo Nicolás Suárez de Tangil
Descripción: Se debe analizar los artefactos ofuscados presentes en una aplicación.	
Estabilidad: Alta	Impacto de estabilidad: Alto

Tabla 6: RU02. Análisis de código ofuscado.

RU03	
Nombre: Análisis dinámico automático	Fuente: Guillermo Nicolás Suárez de Tangil
Descripción: Se debe analizar dinámicamente la aplicación de manera automática.	
Estabilidad: Alta	Impacto de estabilidad: Alto

Tabla 7: RU03. Análisis dinámico automático.

RU04	
Nombre: Análisis estático automático	Fuente: Guillermo Nicolás Suárez de Tangil
Descripción: Se debe analizar estáticamente la aplicación de manera automática.	
Estabilidad: Alta	Impacto de estabilidad: Alto

Tabla 8: RU04. Análisis estático automático.

RU05	
Nombre: Generación automática de caso de prueba	Fuente: Guillermo Nicolás Suárez de Tangil
Descripción: Se debe producir un caso de prueba por cada artefacto ofuscado encontrado, de manera automática.	
Estabilidad: Alta	Impacto de estabilidad: Alto

Tabla 9: RU05. Generación automática de caso de prueba

RU06	
Nombre: Fichero de resultados	Fuente: Guillermo Nicolás Suárez de Tangil
Descripción: Se debe recoger el resultado de los casos de prueba realizados en un fichero de resultados.	
Estabilidad: Alta	Impacto de estabilidad: Bajo

Tabla 10: RU06. Fichero de resultados.

RU07	
Nombre: Análisis estático: cadenas ofuscadas	Fuente: Guillermo Nicolás Suárez de Tangil
Descripción: El sistema debe ser capaz de identificar las cadenas ofuscadas de la aplicación.	
Estabilidad: Alta	Impacto de estabilidad: Bajo

Tabla 11: RU07. Análisis estático: cadenas ofuscadas.

RU08	
Nombre: Análisis dinámico: cadenas ofuscadas	Fuente: Guillermo Nicolás Suárez de Tangil
Descripción: El sistema debe ser capaz de realizar un análisis dinámico de cada cadena ofuscada modificada.	
Estabilidad: Alta	Impacto de estabilidad: Bajo

Tabla 12: RU08. Análisis dinámico: cadenas ofuscadas.

RU09	
Nombre: Casos de prueba: cadenas ofuscadas	Fuente: Guillermo Nicolás Suárez de Tangil
Descripción: El sistema debe realizar, de manera automática, un caso de prueba por cada cadena ofuscada identificada en la aplicación.	
Estabilidad: Alta	Impacto de estabilidad: Bajo

Tabla 13: RU09. Casos de prueba: cadenas ofuscadas.

RU10	
Nombre: Análisis estático: <i>assets</i>	Fuente: Guillermo Nicolás Suárez de Tangil
Descripción: El sistema debe identificar los ficheros de recursos presentes en la carpeta <i>assets</i> dentro de la aplicación.	
Estabilidad: Alta	Impacto de estabilidad: Bajo

Tabla 14: RU10. Análisis estático: *assets*.

RU11	
Nombre: Análisis dinámico: <i>assets</i>	Fuente: Guillermo Nicolás Suárez de Tangil
Descripción: El sistema debe realizar una ejecución dinámica de la aplicación con cada uno de los ficheros <i>assets</i> modificados.	
Estabilidad: Alta	Impacto de estabilidad: Bajo

Tabla 15: RU11. Análisis dinámico: *assets*.

RU12	
Nombre: Casos de prueba: <i>assets</i>	Fuente: Guillermo Nicolás Suárez de Tangil
Descripción: El sistema debe realizar, de manera automática, un caso de prueba por cada fichero <i>asset</i> identificado en la aplicación.	
Estabilidad: Alta	Impacto de estabilidad: Bajo

Tabla 16: RU12. Casos de prueba: *assets*.

RU13	
Nombre: Tiempo de ejecución de análisis dinámico por defecto	Fuente: Guillermo Nicolás Suárez de Tangil
Descripción: El sistema debe de establecer un tiempo de ejecución del análisis dinámico de manera automática, si el usuario no ha indicado dicho tiempo.	
Estabilidad: Media	Impacto de estabilidad: Bajo

Tabla 17: RU13. Tiempo de ejecución de análisis dinámico por defecto

RU14	
Nombre: Selección del tiempo de ejecución de análisis dinámico	Fuente: Guillermo Nicolás Suárez de Tangil
Descripción: El tiempo de ejecución del análisis dinámico debe ser el elegido por el usuario.	
Estabilidad: Alta	Impacto de estabilidad: Medio

Tabla 18: RU14. Selección del tiempo de ejecución de análisis dinámico.

RU15	
Nombre: Análisis estático: condicionales inversos	Fuente: Guillermo Nicolás Suárez de Tangil
Descripción: El sistema debe identificar todos los condicionales presentes en el código y modificarlos a su condición inversa.	
Estabilidad: Alta	Impacto de estabilidad: Bajo

Tabla 19: RU15. Análisis estático: condicionales inversos.

RU16	
Nombre: Análisis dinámico: condicionales inversos	Fuente: Guillermo Nicolás Suárez de Tangil
Descripción: El sistema debe realizar una ejecución dinámica de la aplicación con los condicionales modificados a su condición inversa.	
Estabilidad: Alta	Impacto de estabilidad: Bajo

Tabla 20: RU16. Análisis dinámico: condicionales inversos.

RU17	
Nombre: Caso de prueba: condicionales inversos	Fuente: Guillermo Nicolás Suárez de Tangil
Descripción: Se debe realizar un caso de prueba en el que todos los condicionales presentes en el código de la aplicación se hayan modificado por su condición inversa.	
Estabilidad: Alta	Impacto de estabilidad: Medio

Tabla 21: RU17. Caso de prueba: condicionales inversos.

RU18	
Nombre: Análisis estático: condicionales a true	Fuente: Guillermo Nicolás Suárez de Tangil
Descripción: El sistema debe identificar todos los condicionales presentes en el código y modificar el código de tal manera que las condiciones siempre sean verdaderas.	
Estabilidad: Alta	Impacto de estabilidad: Bajo

Tabla 22: RU18. Análisis estático: condicionales a true.

RU19	
Nombre: Análisis dinámico: condicionales a true	Fuente: Guillermo Nicolás Suárez de Tangil
Descripción: El sistema debe realizar una ejecución dinámica de la aplicación con los condicionales modificados para que siempre se cumplan.	
Estabilidad: Alta	Impacto de estabilidad: Bajo

Tabla 23: RU19. Análisis dinámico: condicionales a true.

RU20	
Nombre: Caso de prueba: condicionales a true	Fuente: Guillermo Nicolás Suárez de Tangil
Descripción: Se debe realizar un caso de prueba en el que todos los condicionales presentes en el código de la aplicación se hayan modificado de tal manera para que su condición se cumpla.	
Estabilidad: Alta	Impacto de estabilidad: Medio

Tabla 24: RU20. Caso de prueba: condicionales a true

RU21	
Nombre: Análisis estático: condicionales a false	Fuente: Guillermo Nicolás Suárez de Tangil
Descripción: El sistema debe identificar todos los condicionales presentes en el código y modificar el código de tal manera que las condiciones siempre sean falsas.	
Estabilidad: Alta	Impacto de estabilidad: Bajo

Tabla 25: RU21. Análisis estático: condicionales a false.

RU22	
Nombre: Análisis dinámico: condicionales a false	Fuente: Guillermo Nicolás Suárez de Tangil
Descripción: El sistema debe realizar una ejecución dinámica de la aplicación con los condicionales modificados para que no se cumplan nunca.	
Estabilidad: Alta	Impacto de estabilidad: Bajo

Tabla 26: RU22. Análisis dinámico: condicionales a false.

RU23	
Nombre: Caso de prueba: condicionales a false	Fuente: Guillermo Nicolás Suárez de Tangil
Descripción: Se debe realizar un caso de prueba en el que todos los condicionales presentes en el código de la aplicación se hayan modificado de tal manera para que su condición no se cumpla.	
Estabilidad: Alta	Impacto de estabilidad: Medio

Tabla 27: RU23. Caso de prueba: condicionales a false.

RU24	
Nombre: Análisis estático: condicionales aleatorios	Fuente: Guillermo Nicolás Suárez de Tangil
Descripción: El sistema debe identificar todos los condicionales presentes en el código y modificar el código de tal manera que sea sustituida por una condición aleatoria.	
Estabilidad: Alta	Impacto de estabilidad: Bajo

Tabla 28: RU24. Análisis estático: condicionales aleatorios.

RU25	
Nombre: Análisis dinámico: condicionales aleatorios	Fuente: Guillermo Nicolás Suárez de Tangil
Descripción: El sistema debe realizar una ejecución dinámica de la aplicación con los condicionales originales sustituidos por condicionales aleatorios.	
Estabilidad: Alta	Impacto de estabilidad: Bajo

Tabla 29: RU25. Análisis dinámico: condicionales aleatorios

RU26	
Nombre: Caso de prueba: condicionales aleatorios	Fuente: Guillermo Nicolás Suárez de Tangil
Descripción: Se debe realizar un caso de prueba en el que todos los condicionales presentes en el código de la aplicación se hayan modificado presentando una nueva condición aleatoria.	
Estabilidad: Alta	Impacto de estabilidad: Medio

Tabla 30: RU26. Caso de prueba: condicionales aleatorios.

RU27	
Nombre: Análisis estático: llamadas a API criptográfica	Fuente: Guillermo Nicolás Suárez de Tangil
Descripción: El sistema debe identificar todas las llamadas a APIs criptográficas que se realizan en el código, modificándolas de tal manera que el resultado obtenido al cifrar o descifrar sea completamente aleatorio.	
Estabilidad: Alta	Impacto de estabilidad: Bajo

Tabla 31: RU27. Análisis estático: llamadas a API criptográfica.

RU28	
Nombre: Análisis dinámico: llamadas a API criptográfica	Fuente: Guillermo Nicolás Suárez de Tangil
Descripción: El sistema debe realizar una ejecución dinámica de la aplicación por cada una de las llamadas a APIs criptográficas modificadas.	
Estabilidad: Alta	Impacto de estabilidad: Bajo

Tabla 32: RU28. Análisis dinámico: llamadas a API criptográfica.

RU29	
Nombre: Caso de prueba: llamadas a API criptográfica	Fuente: Guillermo Nicolás Suárez de Tangil
Descripción: Se debe realizar un caso de prueba por cada llamada a métodos de cifrado y descifrado presentes en el código de la aplicación.	
Estabilidad: Alta	Impacto de estabilidad: Medio

Tabla 33: RU29. Caso de prueba: llamadas a API criptográfica.

RU30	
Nombre: Tester automático de interfaz	Fuente: Guillermo Nicolás Suárez de Tangil
Descripción: Se debe utilizar una herramienta para testear automáticamente la interfaz de usuario de la aplicación durante el análisis dinámico.	
Estabilidad: Alta	Impacto de estabilidad: Medio

Tabla 34: RU30. Tester automático de interfaz.

RU31	
Nombre: Optimización del número de casos de prueba	Fuente: Guillermo Nicolás Suárez de Tangil
Descripción: Se debe realizar un mecanismo que permita optimizar el número de casos de prueba resultantes.	
Estabilidad: Alta	Impacto de estabilidad: Medio

Tabla 35: RU31. Optimización del número de casos de prueba.

RU32	
Nombre: Definición de métrica para la detección de <i>malware</i>	Fuente: Guillermo Nicolás Suárez de Tangil
Descripción: Se debe proporcionar una métrica que permita identificar si la ejecución de un caso de prueba es considerada ordinaria o maliciosa.	
Estabilidad: Alta	Impacto de estabilidad: Medio

Tabla 36: RU32. Definición de métrica para la detección de *malware*

En la Tabla 37 se presenta la matriz de trazabilidad que relaciona a los casos de uso explicados anteriormente con los requisitos de usuario definidos:

	CU1: 1 Analizar aplicación	CU2: Elección del tiempo de prueba del test case	CU3: Visualización de resultados
RU01	X	X	X
RU02	X	X	X
RU03	X	X	X
RU04	X	X	X
RU05	X	X	X
RU06	X	X	X
RU07	X	X	X
RU08	X	X	X
RU09	X	X	X
RU10	X	X	X
RU11	X	X	X
RU12	X	X	X
RU13	X		X
RU14		X	X
RU15	X	X	X
RU16	X	X	X
RU17	X	X	X
RU18	X	X	X
RU19	X	X	X
RU20	X	X	X
RU21	X	X	X
RU22	X	X	X
RU23	X	X	X
RU24	X	X	X
RU25	X	X	X
RU26	X	X	X
RU27	X	X	X

RU28	X	X	X
RU29	X	X	X
RU30	X	X	X
RU31	X	X	X
RU32	X	X	X

Tabla 37: Matriz de trazabilidad: CU-RU.

2.4.3 REQUISITOS SOFTWARE

En esta sección se van a definir los requisitos software, tanto funcionales como no funcionales, del sistema.

2.4.3.1 REQUISITOS FUNCIONALES

RSF01	
Nombre: Identificación de cadenas ofuscadas	Fuente: Mario Herreros Díaz
Descripción: El sistema debe identificar las cadenas de texto ofuscadas de la aplicación a analizar.	
Estabilidad: Alta	Impacto de estabilidad: Alta

Tabla 38: RSF01. Identificación de cadenas ofuscadas.

RSF02	
Nombre: Mutación de cadenas ofuscadas	Fuente: Mario Herreros Díaz
Descripción: El sistema debe modificar las cadenas ofuscadas identificadas.	
Estabilidad: Alta	Impacto de estabilidad: Alta

Tabla 39: RSF02. Mutación de cadenas ofuscadas.

RSF03	
Nombre: Identificación de ficheros <i>assets</i>	Fuente: Mario Herreros Díaz
Descripción: El sistema debe identificar los ficheros <i>assets</i> presentes en la aplicación.	
Estabilidad: Alta	Impacto de estabilidad: Alta

Tabla 40: RSF03. Identificación de ficheros *assets*.

RSF04	
Nombre: Mutación de ficheros <i>assets</i>	Fuente: Mario Herreros Díaz
Descripción: El sistema debe modificar los ficheros <i>assets</i> identificados.	
Estabilidad: Alta	Impacto de estabilidad: Alto

Tabla 41: RSF04. Mutación de ficheros *assets*.

RSF05	
Nombre: Descompilación de la aplicación	Fuente: Mario Herreros Díaz
Descripción: El sistema debe descompilar la aplicación a analizar.	
Estabilidad: Alta	Impacto de estabilidad: Bajo

Tabla 42: RSF05. Descompilación de la aplicación.

RSF06	
Nombre: Compilación de la aplicación	Fuente: Mario Herreros Díaz
Descripción: El sistema debe compilar la aplicación a analizar.	
Estabilidad: Alta	Impacto de estabilidad: Bajo

Tabla 43: RSF06. Compilación de la aplicación.

RSF07	
Nombre: Caso de prueba de la aplicación sin modificar	Fuente: Mario Herreros Díaz
Descripción: El sistema debe realizar un caso de prueba de la aplicación sin que haya modificado ningún elemento.	
Estabilidad: Alta	Impacto de estabilidad: Bajo

Tabla 44: RSF07. Caso de prueba de la aplicación sin modificar.

RSF08	
Nombre: Caso de prueba: modificación string	Fuente: Mario Herreros Díaz
Descripción: El sistema debe realizar un caso de prueba de la aplicación de cada string ofuscado.	
Estabilidad: Alta	Impacto de estabilidad: Medio

Tabla 45: RSF08. Caso de prueba: modificación string.

RSF09	
Nombre: Caso de prueba: modificación fichero <i>assets</i>	Fuente: Mario Herreros Díaz
Descripción: El sistema debe realizar un caso de prueba de la aplicación de cada fichero presente en la carpeta <i>assets</i> .	
Estabilidad: Alta	Impacto de estabilidad: Medio

Tabla 46: RSF09. Caso de prueba: modificación fichero *assets*.

RSF10	
Nombre: Creación AVD	Fuente: Mario Herreros Díaz
Descripción: El sistema debe crear un <i>Android Virtual Device</i> nuevo para cada prueba.	
Estabilidad: Alta	Impacto de estabilidad: Bajo

Tabla 47: RSF10. Creación AVD.

RSF11	
Nombre: Eliminación AVD	Fuente: Mario Herreros Díaz
Descripción: El sistema debe eliminar cada <i>Android Virtual Device</i> utilizado en cada caso de prueba.	
Estabilidad: Alta	Impacto de estabilidad: Bajo

Tabla 48: RSF11. Eliminación AVD.

RSF12	
Nombre: Ejecución caso de prueba sobre DroidBox	Fuente: Mario Herreros Díaz
Descripción: El sistema debe ejecutar cada caso de prueba sobre el programa DroidBox.	
Estabilidad: Alta	Impacto de estabilidad: Medio

Tabla 49: RSF12. Ejecución caso de prueba sobre DroidBox.

RSF13	
Nombre: Contador de operaciones de lecturas	Fuente: Mario Herreros Díaz
Descripción: El sistema debe calcular el total de operaciones de lectura producidas durante el análisis dinámico.	
Estabilidad: Media	Impacto de estabilidad: Bajo

Tabla 50: RSF13. Contador de operaciones de lecturas.

RSF14	
Nombre: Contador de operaciones de escritura	Fuente: Mario Herreros Díaz
Descripción: El sistema debe calcular el total de operaciones de escritura producidas durante el análisis dinámico.	
Estabilidad: Media	Impacto de estabilidad: Bajo

Tabla 51: RSF14. Contador de operaciones de escritura.

RSF15	
Nombre: Contador de conexiones entrantes	Fuente: Mario Herreros Díaz
Descripción: El sistema debe calcular el total de conexiones entrantes producidas durante el análisis dinámico.	
Estabilidad: Media	Impacto de estabilidad: Bajo

Tabla 52: RSF15. Contador de conexiones entrantes.

RSF16	
Nombre: Contador de conexiones salientes	Fuente: Mario Herreros Díaz
Descripción: El sistema debe calcular el total de conexiones salientes producidas durante el análisis dinámico.	
Estabilidad: Media	Impacto de estabilidad: Bajo

Tabla 53: RSF16. Contador de conexiones salientes.

RSF17	
Nombre: Contador de conexiones abiertas	Fuente: Mario Herreros Díaz
Descripción: El sistema debe calcular el total de conexiones abiertas producidas durante el análisis dinámico.	
Estabilidad: Media	Impacto de estabilidad: Bajo

Tabla 54: RSF17. Contador de conexiones abiertas.

RSF18	
Nombre: Contador de fugas de información	Fuente: Mario Herreros Díaz
Descripción: El sistema debe calcular el total de información filtrada durante el análisis dinámico.	
Estabilidad: Baja	Impacto de estabilidad: Baja

Tabla 55: RSF18. Contador de fugas de información.

RSF19	
Nombre: Tiempo medio: lecturas en disco	Fuente: Mario Herreros Díaz
Descripción: El sistema debe calcular el tiempo medio de operaciones de lectura en disco.	
Estabilidad: Media	Impacto de estabilidad: Medio

Tabla 56: RSF19. Tiempo medio: lecturas en disco.

RSF20	
Nombre: Tiempo medio: escrituras en disco	Fuente: Mario Herreros Díaz
Descripción: El sistema debe calcular el tiempo medio de operaciones de escritura en disco.	
Estabilidad: Media	Impacto de estabilidad: Medio

Tabla 57: RSF20. Tiempo medio: escrituras en disco.

RSF21	
Nombre: Contador de operaciones criptográficas	Fuente: Mario Herreros Díaz
Descripción: El sistema debe calcular el total de operaciones criptográficas producidas durante el análisis dinámico.	
Estabilidad: Baja	Impacto de estabilidad: Bajo

Tabla 58: RSF21. Contador de operaciones criptográficas.

RSF22	
Nombre: Contador de operaciones totales	Fuente: Mario Herreros Díaz
Descripción El sistema debe calcular el total de operaciones recogidas por DroidBox producidas durante el análisis dinámico.	
Estabilidad: Baja	Impacto de estabilidad: Bajo

Tabla 59: RSF22. Contador de operaciones totales.

RSF23	
Nombre: Generación de fichero de resultados	Fuente: Mario Herreros Díaz
Descripción El sistema debe generar un fichero con los resultados obtenidos en cada caso de prueba.	
Estabilidad: Alta	Impacto de estabilidad: Medio

Tabla 60: RSF23. Generación de fichero de resultados.

RSF24	
Nombre: Identificación de sentencias condicionales	Fuente: Mario Herreros Díaz
Descripción El sistema debe identificar todas las sentencias condicionales presentes en el código fuente de la aplicación.	
Estabilidad: Alta	Impacto de estabilidad: Medio

Tabla 61: RSF24. Identificación de sentencias condicionales.

RSF25	
Nombre: Modificación de sentencias condicionales: al inverso	Fuente: Mario Herreros Díaz
Descripción El sistema debe sustituir cada sentencia condicional por su condición inversa.	
Estabilidad: Alta	Impacto de estabilidad: Medio

Tabla 62: RSF25: Modificación de sentencias condicionales: al inverso.

RSF26	
Nombre: Caso de prueba: inversión de sentencias condicionales	Fuente: Mario Herreros Díaz
Descripción El sistema debe realizar un caso de prueba con todas las condiciones del código fuente invertidas.	
Estabilidad: Alta	Impacto de estabilidad: Medio

Tabla 63: RSF26. Caso de prueba: inversión de sentencias condicionales.

RSF27	
Nombre: Modificación de sentencias condicionales: true	Fuente: Mario Herreros Díaz
Descripción El sistema debe modificar cada condición del código de la aplicación para que su resultado sea cierto.	
Estabilidad: Alta	Impacto de estabilidad: Medio

Tabla 64: RSF27. Modificación de sentencias condicionales: true.

RSF28	
Nombre: Caso de prueba: true	Fuente: Mario Herreros Díaz
Descripción El sistema debe realizar un caso de prueba con todas las condiciones del código modificadas para que siempre se cumplan	
Estabilidad: Alta	Impacto de estabilidad: Medio

Tabla 65: RSF28. Caso de prueba: true.

RSF29	
Nombre: Modificación de sentencias condicionales: false	Fuente: Mario Herreros Díaz
Descripción El sistema debe modificar cada condición del código de la aplicación para que su resultado sea falso.	
Estabilidad: Alta	Impacto de estabilidad: Medio

Tabla 66: RSF29. Modificación de sentencias condicionales: false.

RSF30	
Nombre: Caso de prueba: false	Fuente: Mario Herreros Díaz
Descripción El sistema debe realizar un caso de prueba con todas las condiciones del código modificadas para que nunca se cumplan.	
Estabilidad: Alta	Impacto de estabilidad: Medio

Tabla 67: RSF30. Caso de prueba: false.

RSF31	
Nombre: Modificación de sentencias condicionales: aleatorio	Fuente: Mario Herreros Díaz
Descripción El sistema debe sustituir cada condición por una condición aleatoria.	
Estabilidad: Alta	Impacto de estabilidad: Medio

Tabla 68: RSF31. Modificación de sentencias condicionales: aleatorio.

RSF32	
Nombre: Caso de prueba: condición aleatoria	Fuente: Mario Herreros Díaz
Descripción El sistema debe realizar un caso de prueba con todas las condiciones del código modificadas por condiciones aleatorias.	
Estabilidad: Alta	Impacto de estabilidad: Medio

Tabla 69: RSF32. Caso de prueba: condición aleatoria.

RSF33	
Nombre: Identificación de llamadas a APIs criptográficas	Fuente: Mario Herreros Díaz
Descripción El sistema debe identificar todas las llamadas que se realicen en el código de la aplicación a APIs criptográficas.	
Estabilidad: Alta	Impacto de estabilidad: Medio

Tabla 70: RSF33. Identificación de llamadas a APIs criptográficas.

RSF34	
Nombre: Modificación de llamadas a APIs criptográficas	Fuente: Mario Herreros Díaz
Descripción El sistema debe modificar las llamadas a APIs criptográficas para modificar el resultado de operaciones de cifrado y descifrado por un conjunto de bytes aleatorios.	
Estabilidad: Alta	Impacto de estabilidad: Medio

Tabla 71: RSF34. Modificación de llamadas a APIs criptográficas.

RSF35	
Nombre: Caso de prueba: modificación de llamada criptográfica	Fuente: Mario Herreros Díaz
Descripción El sistema debe realizar un caso de prueba por cada llamada criptográfica modificada.	
Estabilidad: Alta	Impacto de estabilidad: Medio

Tabla 72: RSF35. Caso de prueba: modificación de llamada criptográfica.

RSF36	
Nombre: Generación de un tester automático	Fuente: Mario Herreros Díaz
Descripción El sistema debe generar un tester que maneje de manera automática la interfaz de la aplicación durante el análisis utilizando DroidBox.	
Estabilidad: Alta	Impacto de estabilidad: Medio

Tabla 73: RSF36. Generación de un tester automático.

RSF37	
Nombre: Contador UDP	Fuente: Mario Herreros Díaz
Descripción El sistema debe calcular el total de paquetes UDP circulantes durante el análisis dinámico.	
Estabilidad: Baja	Impacto de estabilidad: Bajo

Tabla 74: RSF37. Contador UDP.

RSF38	
Nombre: Contador TCP	Fuente: Mario Herreros Díaz
Descripción El sistema debe calcular el total de paquetes TCP circulantes durante el análisis dinámico.	
Estabilidad: Baja	Impacto de estabilidad: Bajo

Tabla 75: RSF38. Contador TCP.

RSF39	
Nombre: Contador DNS	Fuente: Mario Herreros Díaz
Descripción El sistema debe calcular el total de peticiones DNS producidas durante el análisis dinámico.	
Estabilidad: Baja	Impacto de estabilidad: Bajo

Tabla 76: RSF39. Contador DNS.

RSF40	
Nombre: Contador de URLs cifradas	Fuente: Mario Herreros Díaz
Descripción El sistema debe calcular el total de URLs, presentes en las peticiones DNS, que estén cifradas.	
Estabilidad: Baja	Impacto de estabilidad: Bajo

Tabla 77: RSF40. Contador de URLs cifradas.

RSF41	
Nombre: Contador de URLs en claro	Fuente: Mario Herreros Díaz
Descripción El sistema debe calcular el total de URLs, presentes en las peticiones DNS, que estén en texto plano.	
Estabilidad: Baja	Impacto de estabilidad: Bajo

Tabla 78: RSF41. Contador de URLs en claro.

RSF42	
Nombre: Contador de HTTP POST	Fuente: Mario Herreros Díaz
Descripción El sistema debe calcular el total de peticiones HTTP POST producidas durante la ejecución de la aplicación.	
Estabilidad: Baja	Impacto de estabilidad: Bajo

Tabla 79: RSF42. Contador de HTTP POST.

RSF43	
Nombre: Contador de HTTP GET	Fuente: Mario Herreros Díaz
Descripción El sistema debe calcular el total de peticiones HTTP POST producidas durante la ejecución de la aplicación.	
Estabilidad: Baja	Impacto de estabilidad: Bajo

Tabla 80: RSF43. Contador de HTTP GET.

RSF44	
Nombre: Contador de POST cifrado	Fuente: Mario Herreros Díaz
Descripción El sistema debe calcular el total de peticiones HTTP POST cuya información enviada esté cifrada.	
Estabilidad: Baja	Impacto de estabilidad: Bajo

Tabla 81: RSF44. Contador de POST cifrado.

RSF45	
Nombre: Contador de POST en claro	Fuente: Mario Herreros Díaz
Descripción El sistema debe calcular el total de peticiones HTTP POST cuya información enviada esté en texto plano.	
Estabilidad: Baja	Impacto de estabilidad: Bajo

Tabla 82: RSF45. Contador de POST en claro.

RSF46	
Nombre: Contador de GET cifrado	Fuente: Mario Herreros Díaz
Descripción El sistema debe calcular el total de peticiones HTTP GET cuya información enviada esté cifrada.	
Estabilidad: Baja	Impacto de estabilidad: Bajo

Tabla 83: RSF46. Contador de GET cifrado.

RSF47	
Nombre: Contador de GET en claro	Fuente: Mario Herreros Díaz
Descripción El sistema debe calcular el total de peticiones HTTP GET cuya información enviada esté en texto plano.	
Estabilidad: Baja	Impacto de estabilidad: Bajo

Tabla 84: RSF47. Contador de GET en claro.

RSF48	
Nombre: Contador de operaciones <i>dexclassloader</i>	Fuente: Mario Herreros Díaz
Descripción: El sistema debe calcular el total de operaciones <i>dexclassloader</i> producidas durante el análisis dinámico.	
Estabilidad: Media	Impacto de estabilidad: Bajo

Tabla 85: RSF48. Contador de operaciones *dexclassloader*.

RSF49	
Nombre: Contador de servicios iniciados	Fuente: Mario Herreros Díaz
Descripción: El sistema debe calcular el total de servicios activados durante el análisis dinámico.	
Estabilidad: Media	Impacto de estabilidad: Bajo

Tabla 86: RSF49. Contador de servicios iniciados.

RSF50	
Nombre: Mecanismo de optimización del número de casos de prueba	Fuente: Mario Herreros Díaz
Descripción El sistema debe poseer un mecanismo para optimizar el total de número de casos de prueba resultantes.	
Estabilidad: Media	Impacto de estabilidad: Medio

Tabla 87: RSF50. Mecanismo de optimización del número de casos de prueba.

RSF51	
Nombre: Detección de <i>malware</i>	Fuente: Mario Herreros Díaz
Descripción El sistema debe ser capaz de generar una métrica de manera automática que permita considerar, según los resultados obtenidos, si la ejecución del caso de prueba ha sido ordinaria o maliciosa.	
Estabilidad: Alta	Impacto de estabilidad: Medio

Tabla 88: RSF51. Detección de *malware*.

2.4.3.2 REQUISITOS NO FUNCIONALES

RSNF01	
Nombre: Santoku	Fuente: Mario Herreros Díaz
Descripción: El sistema a utilizar durante el desarrollo del sistema debe ser <i>Santoku-linux</i> .	
Estabilidad: Alto	Impacto de estabilidad: Bajo

Tabla 89: RSNF01. Santoku.

RSNF02	
Nombre: <i>ApkTool</i>	Fuente: Guillermo Nicolás Suárez de Tangil
Descripción: Se debe utilizar la herramienta <i>ApkTool</i> para el desarrollo del sistema.	
Estabilidad: Alta	Impacto de estabilidad: Alto

Tabla 90: RSNF02. APKTool

RSNF03	
Nombre: <i>Androguard</i>	Fuente: Guillermo Nicolás Suárez de Tangil
Descripción: Se debe utilizar la herramienta <i>Androguard</i> para el desarrollo del sistema.	
Estabilidad: Alta	Impacto de estabilidad: Alto

Tabla 91: RSNF03. Androguard.

RSNF04	
Nombre: <i>DroidBox</i>	Fuente: Guillermo Nicolás Suárez de Tangil
Descripción: Se debe utilizar la herramienta <i>DroidBox</i> para el desarrollo del sistema.	
Estabilidad: Alta	Impacto de estabilidad: Alto

Tabla 92: RSNF04. DroidBox.

RSNF05	
Nombre: Tiempo de ejecución por defecto	Fuente: Mario Herreros Díaz
Descripción: Se debe establecer un tiempo de ejecución por defecto del análisis dinámico.	
Estabilidad: Medio	Impacto de estabilidad: Bajo

Tabla 93: RSNF05. Tiempo de ejecución por defecto.

RSNF06	
Nombre: Fichero de resultados: hoja de cálculo	Fuente: Guillermo Nicolás Suárez de Tangil
Descripción: El fichero de resultados debe tratarse de una hoja de cálculo.	
Estabilidad: Media	Impacto de estabilidad: Medio

Tabla 94: RSNF06. Fichero de resultados: hoja de cálculo.

RSNF07	
Nombre: Tiempo de ejecución análisis dinámico elegible	Fuente: Mario Herreros Díaz
Descripción: Se debe dar la opcionalidad al usuario de elegir el tiempo de ejecución del análisis dinámico.	
Estabilidad: Medio	Impacto de estabilidad: Bajo

Tabla 95: RSNF07. Tiempo de ejecución análisis dinámico elegible.

RSNF08	
Nombre: <i>Tcpdump</i>	Fuente: Guillermo Nicolás Suárez de Tangil
Descripción: Se debe utilizar la herramienta <i>Tcpdump</i> para filtrar el tráfico de red producido durante el análisis dinámico.	
Estabilidad: Alta	Impacto de estabilidad: Medio

Tabla 96: RSNF08. Tcpdump.

En la Tabla 97 se muestra la matriz de trazabilidad que relaciona los requisitos de usuario definidos anteriormente con los requisitos software, tanto funcionales como no funcionales:

	RU01	RU02	RU03	RU04	RU05	RU06	RU07	RU08	RU09	RU10	RU11	RU12	RU13	RU14	RU15	RU16	RU17	RU18	RU19	RU20	RU21	RU22	RU23	RU24	RU25	RU26	RU27	RU28	RU29	RU30	RU31	RU32
RSF01	X	X		X			X																									
RSF02	X	X							X																							
RSF03	X			X						X																						
RSF04	X											X																				
RSF05	X			X																												
RSF06	X			X																												
RSF07	X				X																											
RSF08	X	X			X				X																							
RSF09	X		X		X							X																				
RSF10			X																													
RSF11			X																													
RSF12	X		X		X			X			X					X			X			X			X			X				
RSF13						X																										
RSF14						X																										
RSF15						X																										
RSF16						X																										
RSF17						X																										
RSF18						X																										
RSF19						X																										
RSF20						X																										
RSF21						X																										
RSF22						X																										
RSF23						X																										
RSF24	X			X											X			X			X			X								
RSF25	X																X															
RSF26	X				X												X															
RSF27	X																			X												
RSF28	X				X															X												
RSF29	X																						X									
RSF30	X				X																		X									
RSF31	X																									X						

[illegible]

Tabla 97: Matriz de trazabilidad: RU-RS.

2.5 ESTUDIO PREVIO

Se ha realizado un estudio previo en el que se han analizado distintas aplicaciones Android infectadas con código malicioso. El *malware* presente en estas aplicaciones utiliza distintas técnicas de ofuscación para llevar a cabo sus acciones malintencionadas. El objetivo de este análisis es estudiar el comportamiento de las aplicaciones tras mutar artefactos ofuscados para poder, a posteriori, diseñar un sistema que permita identificar de forma autónoma qué elementos contienen información oculta mediante técnicas de ofuscación y detectar *payloads* en ellos.

2.5.1 CASO DE PRUEBA 1: ANSERVERBOT

El siguiente caso de prueba se centra en la ofuscación de archivos dentro de la carpeta *assets* de una aplicación Android.

La aplicación a analizar pertenece a la familia *AnserverBot*. Esta familia de *malware* se compone de troyanos, que se caracterizan por emplear técnicas sofisticadas para evadir y frustrar análisis y detecciones de malware (5). En concreto esta aplicación presenta, en su carpeta *assets*, dos archivos denominados *anservera.db* y *anserverb.db*. Estos dos ficheros esconden dos aplicaciones que resultan ser *payloads*.

Este caso de prueba se centrará en modificar el contenido de cada archivo por una secuencia de bytes aleatorios de longitud igual al tamaño del fichero modificado.

ANÁLISIS DEL COMPORTAMIENTO ORDINARIO

En este primer análisis se ha realizado una ejecución de la aplicación durante 120 segundos, sin realizar ninguna modificación, utilizando la herramienta *DroidBox*. Los resultados obtenidos se aprecian en las Ilustraciones 16 y 17:

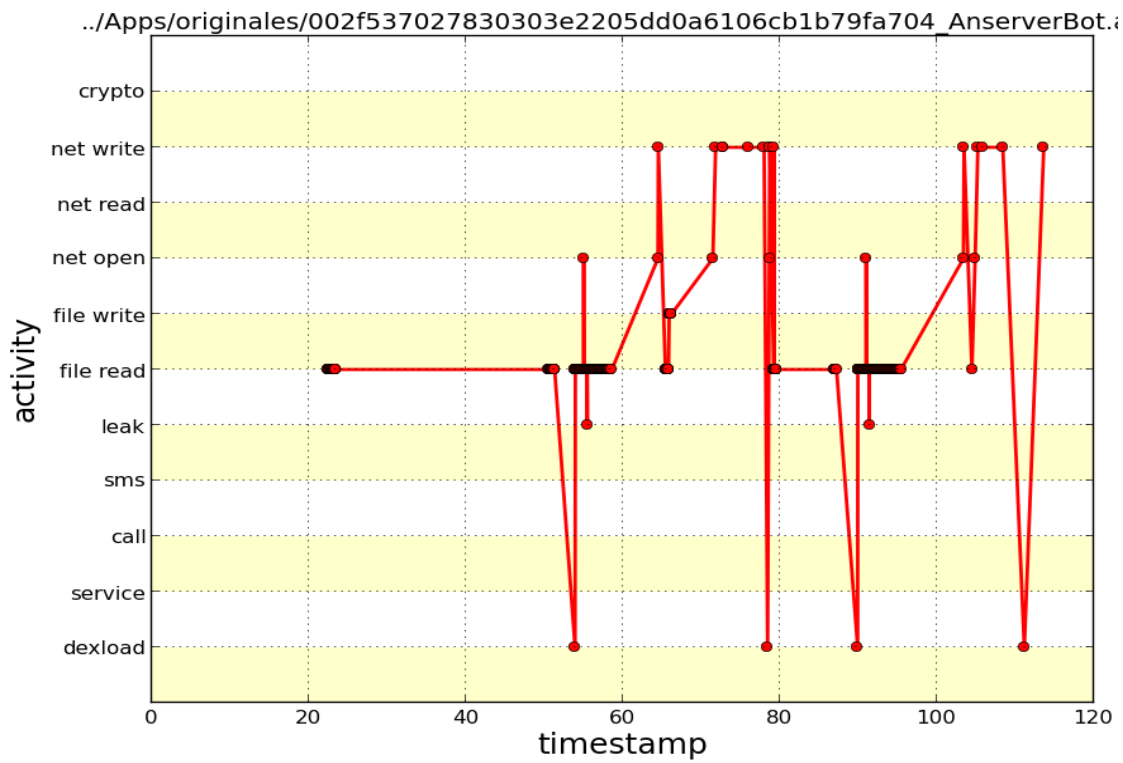


Ilustración 16: TC1-AnserverBot. Comportamiento ordinario: 1.

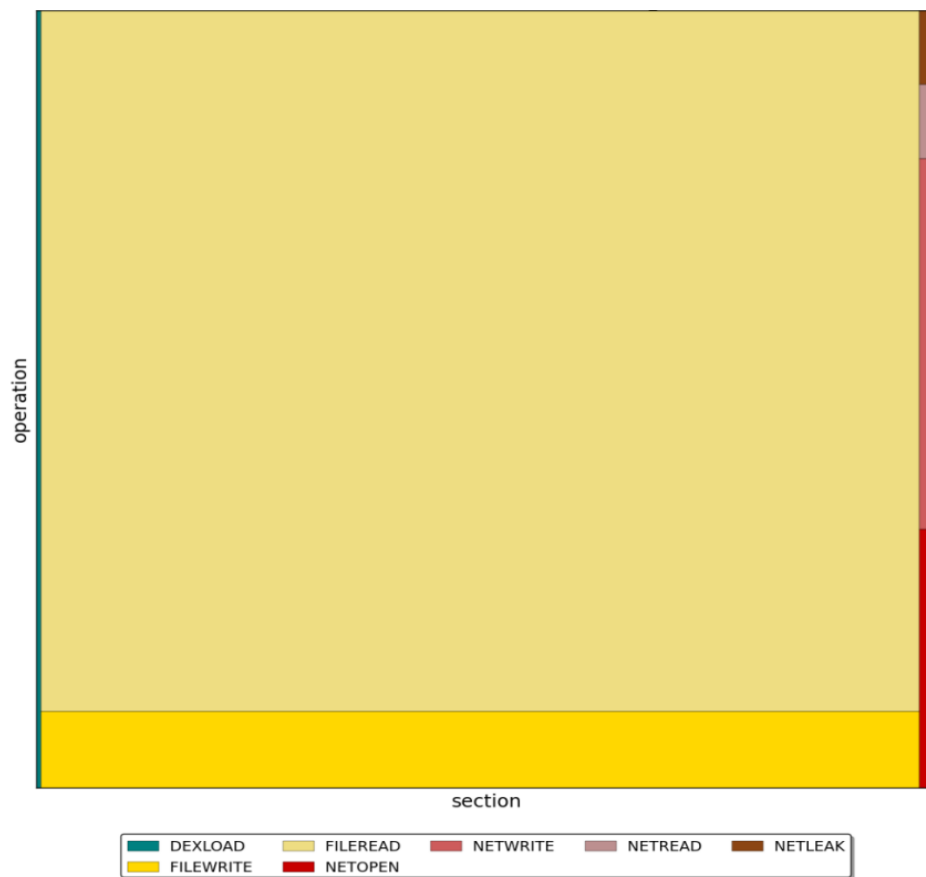


Ilustración 17: TC1-AnserverBot. Comportamiento ordinario: 2

Se han producido operaciones *dexload*, las correspondientes a la instalación de las aplicaciones ofuscadas en los archivos de la carpeta *assets*. También se ha detectado fugas de información, entre operaciones de lectura y escritura de ficheros, así como apertura y escritura de conexiones.

CASO DE PRUEBA

El caso de prueba realizado se resume en:

Introducción	
Identificador	TC1-AnserverBot
Nombre	Carpeta <i>assets</i> : ofuscación de ficheros
Aplicación analizada	Archivo: 002f537027830303e2205dd0a6106cb1b79fa704_AnserverBot.apk MD5: abb5a5c349c372fc24856086a4010ee6 SHA1: 002f537027830303e2205dd0a6106cb1b79fa704
Propósito	Estudiar el comportamiento tras modificar el contenido de ficheros, contenidos en la carpeta <i>assets</i> , que presentan ofuscación.
Actividades	
Ambiente de prueba	<ul style="list-style-type: none"> • Sistema Santoku-Linux. • AVD con versión Android 2.1. • DroidBox instalado. • ApkTool instalado. • Android SDK instalado. • Dex2Jar instalado. • Conexión a Internet.
Acciones	<ol style="list-style-type: none"> 1. Desensamblado de la aplicación a estudiar mediante el uso de ApkTool. 2. Localización e identificación de los archivos, mediante un explorador de archivos, de la carpeta <i>assets</i>: <i>anservera.db</i> y <i>anserverb.db</i>. 3. Modificación, empleando un editor de texto plano, del contenido de cada uno de los archivos por una secuencia de bytes aleatorios. 4. Re-empaquetamiento de la aplicación modificada utilizando ApkTool. 5. Firmar aplicación modificada utilizando Dex2Jar. 6. Ejecución de la aplicación en un sistema AVD versión 2.1 utilizando “Droidbox”. 7. Interacción durante 120 segundos con la UI de la aplicación.
Resultados	

Salida esperada	Contrastes en el comportamiento de la ejecución de la aplicación modificada.
Salida obtenida	La aplicación no ha podido instalar los <i>payloads</i> que escondían los ficheros modificados (no presencia de operaciones <i>dexload</i>).

Tabla 98: TC1-AnserverBot. Carpeta *assets*: ofuscación de ficheros.

En las Ilustraciones 18 y 19 se resumen los resultados obtenidos tras realizar el caso de prueba indicado:

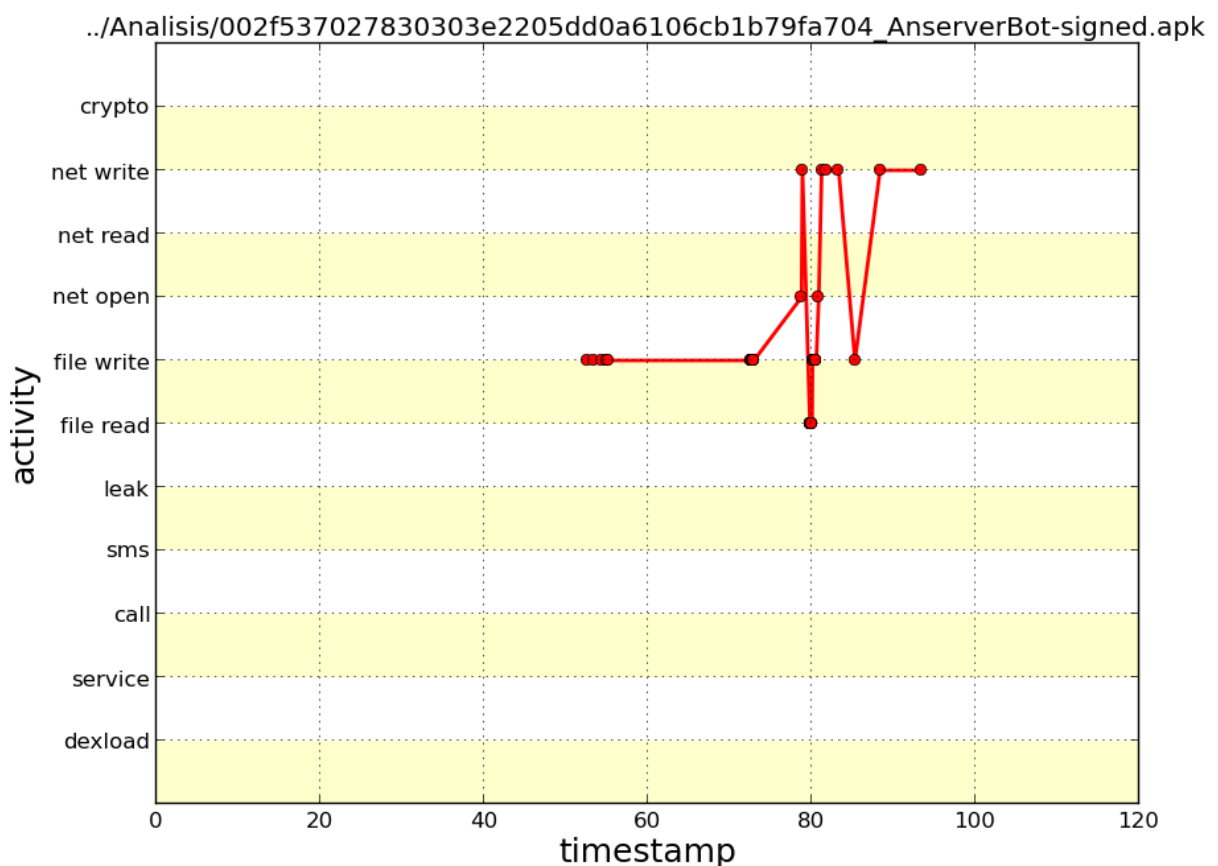


Ilustración 18: TC1-AnserverBot. Resultado final: 1.

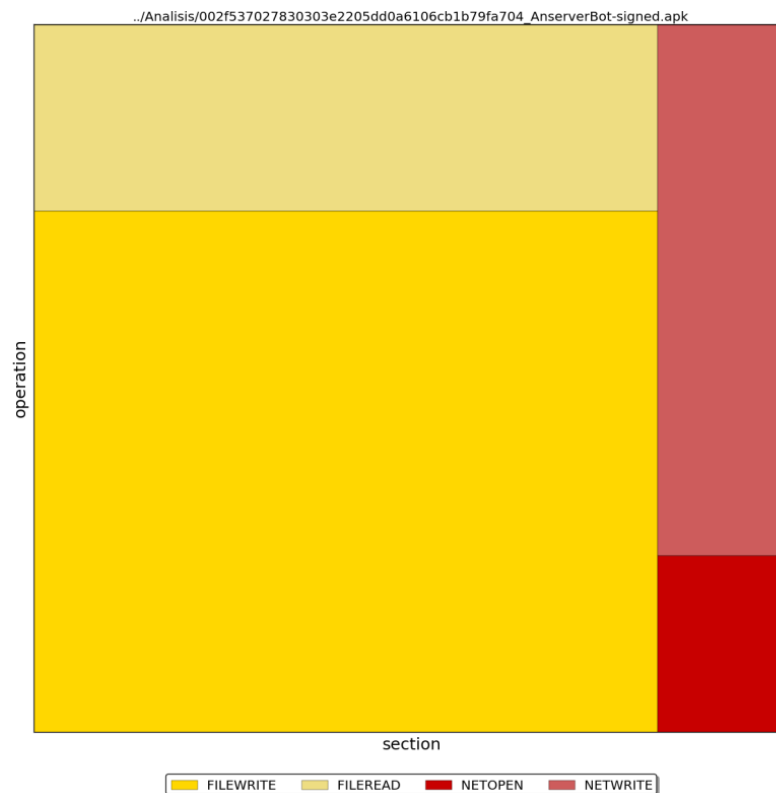


Ilustración 19: TC1-AnserverBot. Resultado final: 2.

Los resultados obtenidos muestran que las aplicaciones que contienen los *payloads* y que se escondían tras los ficheros denominados *anservera.db* y *anserverb.db* no se han podido instalar, ya que no se han producido operaciones del tipo *dexload*. Esto es debido a las modificaciones realizadas en dichos ficheros.

2.5.2 CASO DE PRUEBA 2: DROIDKUNGFU

Este caso de prueba se centra en la presencia de cadenas ofuscadas que son utilizadas por el código malicioso para llevar a cabo sus propósitos.

La aplicación a analizar pertenece a la familia de *malware DroidKungFu*. Este tipo de software malicioso descifra en tiempo de ejecución un *exploit*, incrustado en un fichero denominado *ratc*, acrónimo de *Rage Against the Cage*. Este *exploit* permite al código malintencionado escalar privilegios y poder filtrar información del dispositivo móvil infectado (34).

El caso de prueba consistirá en modificar la clave utilizada para descifrar el *exploit ratc* en tiempo de ejecución, y estudiar su comportamiento.

ANÁLISIS DE COMPORTAMIENTO ORDINARIO

En esta prueba inicial se ha llevado a cabo una ejecución de la aplicación durante 120 segundos, sin realizar ninguna modificación, utilizando la herramienta *DroidBox*. Los resultados obtenidos se aprecian en las Ilustraciones 20 y 21:

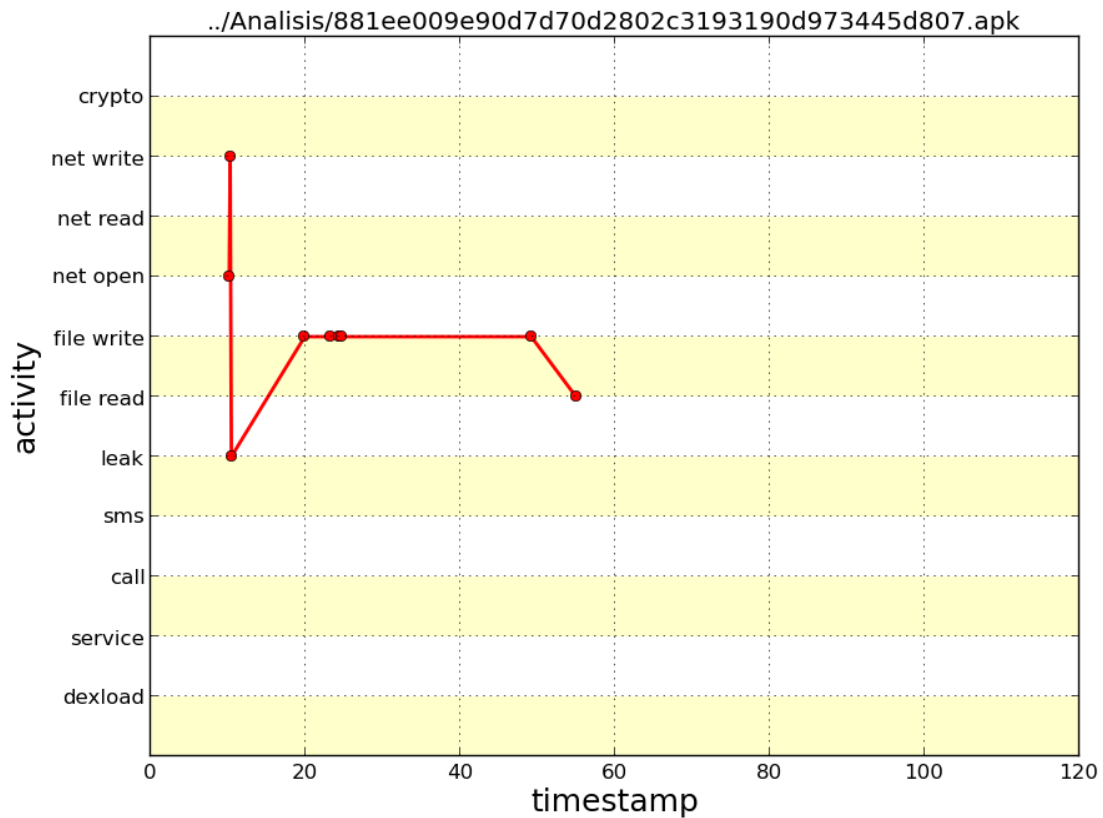


Ilustración 20: TC2-DroidKungFu. Comportamiento ordinario: 1.

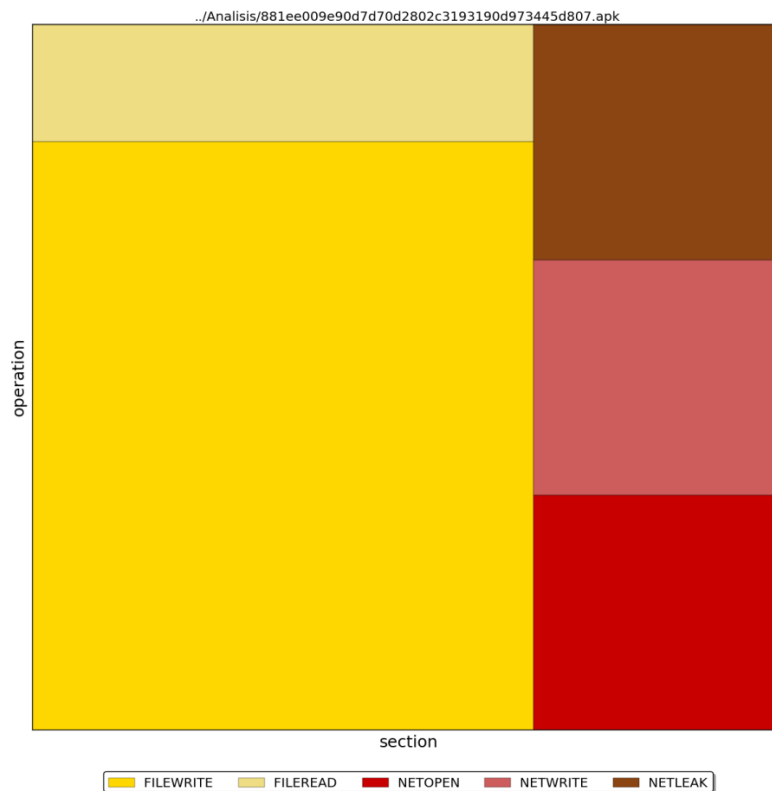


Ilustración 21: TC2-DroidKungFu. Comportamiento ordinario: 2.

Las ilustraciones muestran que la aplicación ha enviado datos a una conexión saliente, filtrando información. La conexión se ha realizado a la siguiente dirección: r2.adwo.com. Con estos datos, se observa que la aplicación infectada ha enviado datos referentes del terminal a dicho servidor de comando y control.

También se han producido distintas operaciones de escritura y lectura de ficheros.

CASO DE PRUEBA

El caso de prueba consiste en:

Introducción	
Identificador	TC2-DroidKungFu
Nombre	Cadena ofuscada
Aplicación analizada	Archivo: 881ee009e90d7d70d2802c3193190d973445d807.apk MD5: 7f5fd7b139e23bed1de5e134dda3b1ca SHA1: 881ee009e90d7d70d2802c3193190d973445d807
Propósito	El fin de este caso de prueba es estudiar el comportamiento tras modificar la clave utilizada para descifrar archivos que almacenan <i>exploits</i> .
Actividades	

Ambiente de prueba	<ul style="list-style-type: none"> • Sistema Santoku-Linux. • AVD con versión Android 2.1. • DroidBox instalado. • ApkTool instalado. • Android SDK instalado. • Dex2Jar instalado. • Conexión a Internet.
Acciones	<ol style="list-style-type: none"> 1. Desensamblado de la aplicación DroidKungFu mediante el uso de APKTool. 2. Identificación de código ofuscado, empleando un editor de texto. En este caso concreto en la clase “Utils” del paquete “com.google.ssearch” se encuentra un array de datos ofuscado. 3. Modificación de dicho array de datos por bytes aleatorios mediante un editor de texto. 4. Re-empaquetamiento de la aplicación modificada utilizando ApkTool. 5. Firmar aplicación modificada utilizando Dex2Jar. 6. Ejecución de la aplicación en un sistema AVD versión 2.1 utilizando “Droidbox”. 7. Interacción durante 120 segundos con la UI de la aplicación.
Resultados	
Salida esperada	Comportamiento distinto al de la ejecución inicial.
Salida obtenida	El malware no ha podido conectarse al servidor r2.adwo.com al que filtraba la información del terminal.

Tabla 99: TC2-DroidKungFu. Cadena ofuscada.

Tras la realización del caso de prueba, los resultados obtenidos se resumen en las Ilustraciones 22 y 23:

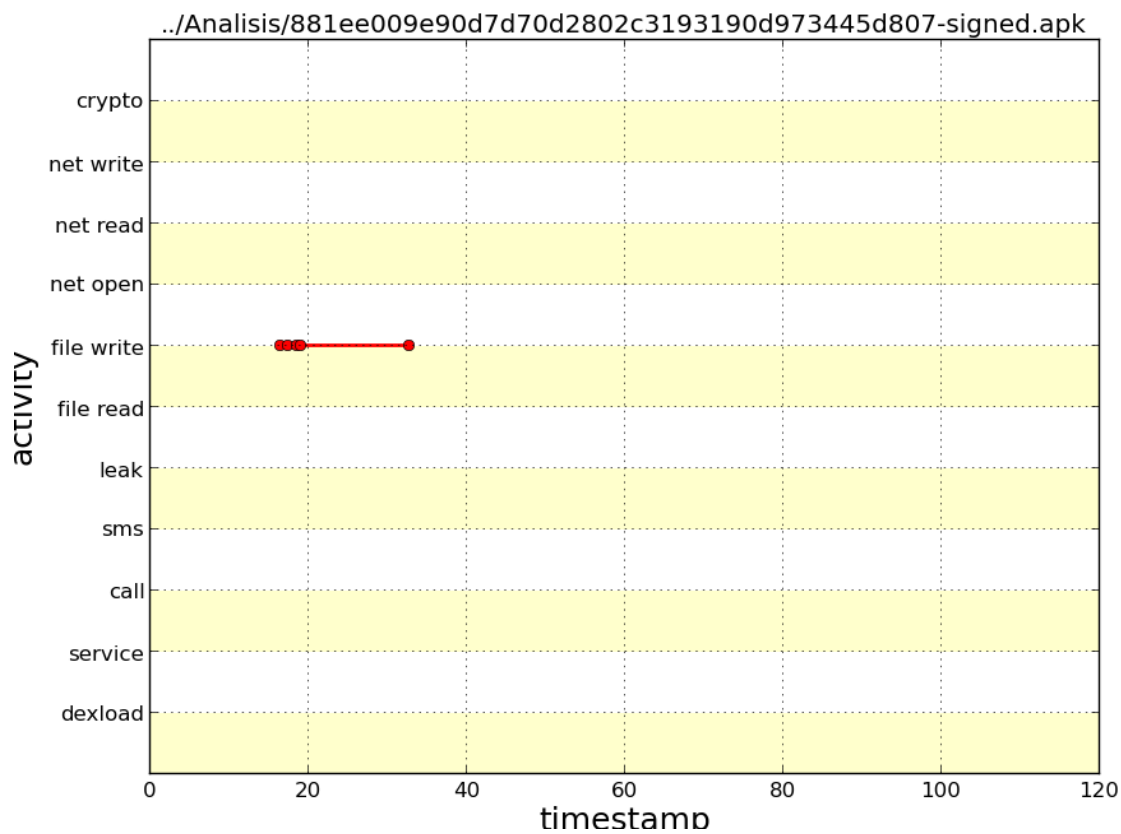


Ilustración 22: TC2-DroidKungFu. Resultado final: 1.

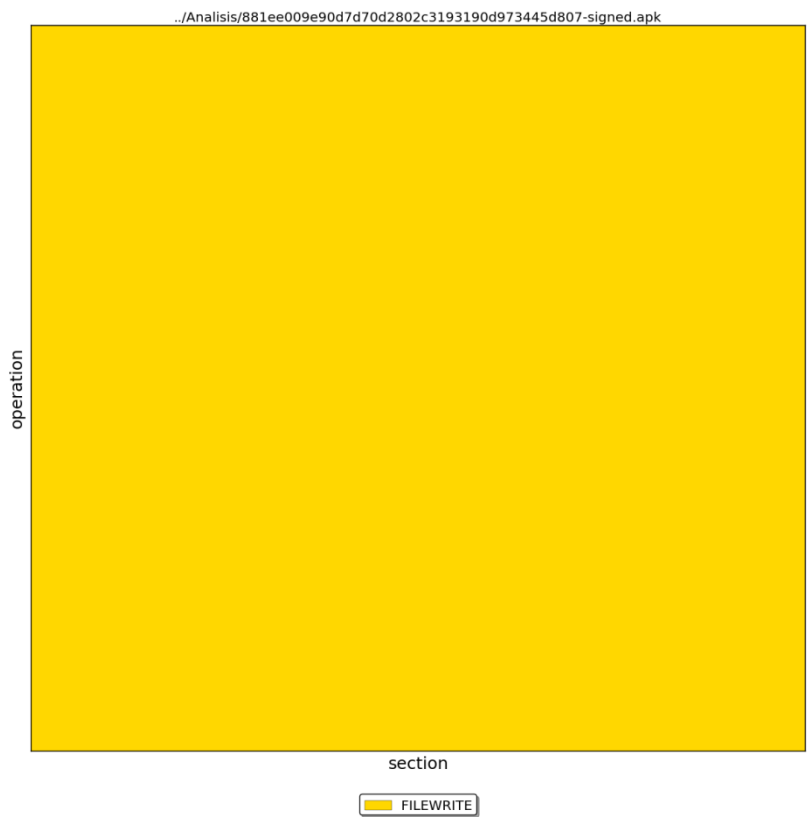


Ilustración 23: TC2-DroidKungFu. Resultado final: 2.

No se ha producida ninguna operación de conexión y no se ha filtrado ni enviado ningún tipo de dato. Esto es debido a que al modificar la clave utilizada para descifrar el fichero *ratc*, que contiene el *exploit* utilizado por el *malware*, el descifrado obtenido no ha sido el esperado. Esto ha impedido al *exploit* realizar su función, incapacitando la funcionalidad del código malintencionado.

2.5.3 CASO DE PRUEBA 3: GINGERMASTER

En este caso de prueba se va a estudiar la ofuscación a través de archivos de imagen.

La familia de *malware* elegida para realizar este estudio ha sido *GingerMaster*. Este tipo de malware ofusca el nombre de ficheros situados en la carpeta *assets*, añadiendo sufijos pertenecientes a archivos de imagen, como puede ser *.png* o *.jpg*. Estos archivos lo que realmente esconden en su interior es código utilizado para escalar privilegios dentro del sistema Android (1).

El objetivo de este caso de prueba consistirá en modificar el contenido de estos ficheros, mutando los *exploits* que contienen, para estudiar el comportamiento de la aplicación infectada.

ANÁLISIS DE COMPORTAMIENTO ORDINARIO

En esta prueba inicial se ha llevado a cabo una ejecución de la aplicación durante 120 segundos, sin realizar ninguna modificación, utilizando la herramienta *DroidBox*. Los resultados obtenidos se aprecian en las Ilustraciones 24 y 25:

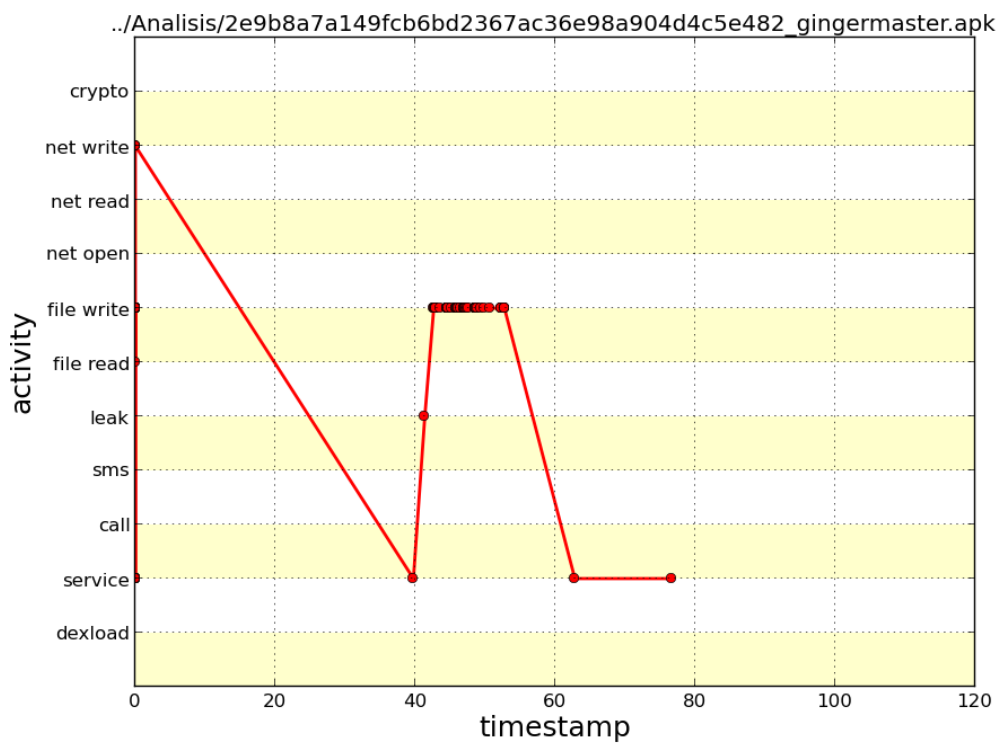


Ilustración 24: TC3-GingerMaster. Comportamiento ordinario: 1.

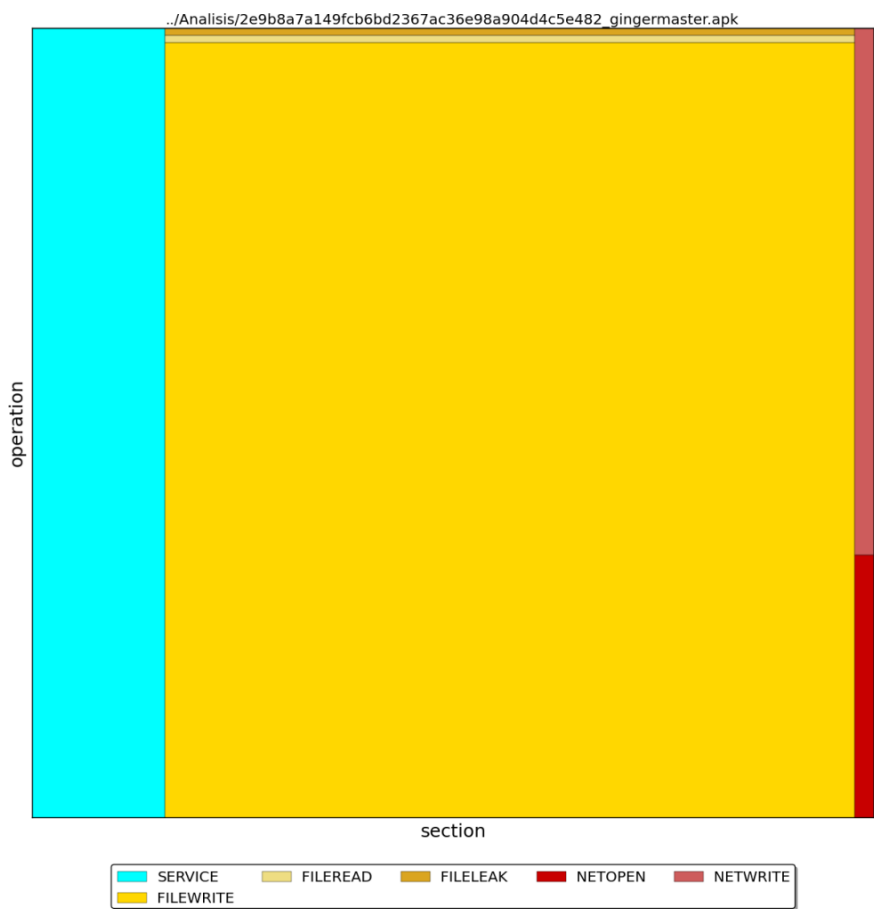


Ilustración 25: TC3-GingerMaster. Comportamiento ordinario: 2.

Se han producido operaciones de escritura en la red junto a filtración de información. También una serie de operaciones de escritura en disco y activación de servicios.

CASO DE PRUEBA

Introducción	
Identificador	TC3-GingerMaster
Nombre	Ofuscación imagen <i>assets</i>
Aplicación analizada	Archivo: 2e9b8a7a149fcb6bd2367ac36e98a904d4c5e482_gingermaster.apk MD5: b0cf27edcb71b055ad20d3e055ef85ca SHA1: 2e9b8a7a149fcb6bd2367ac36e98a904d4c5e482
Propósito	El fin de este caso de prueba es estudiar el comportamiento tras modificar imágenes que contienen código malicioso.
Actividades	
Ambiente de prueba	<ul style="list-style-type: none"> • Sistema Santoku-Linux. • AVD con versión Android 2.1. • DroidBox instalado. • ApkTool instalado. • Android SDK instalado. • Dex2Jar instalado. • Conexión a Internet.
Acciones	<ol style="list-style-type: none"> 1. Desensamblado de la aplicación GingerMaster mediante el uso de APKTool. 2. Identificación de código ofuscado empleando un editor de texto. Para este caso, los ficheros siguientes en la carpeta <i>assets</i>: <i>install.png</i>, <i>installsoft.png</i>, <i>runme.png</i>, <i>gbfm.png</i>. 3. Modificación del contenido de los ficheros descritos por bytes aleatorios empleando un editor de texto. 4. Re-empaquetamiento de la aplicación modificada utilizando ApkTool. 5. Firmar aplicación modificada utilizando Dex2Jar. 6. Ejecución de la aplicación en un sistema AVD versión 2.1 utilizando “Droidbox”. 7. Interacción durante 120 segundos con la UI de la aplicación.
Resultados	
Salida esperada	Comportamiento distinto al de la ejecución inicial.
Salida obtenida	No se ha producido filtrado de información.

Tabla 100: TC3-GingerMaster. Ofuscación imagen asset.

Los resultados obtenidos se resumen en las Ilustraciones 26 y 27:

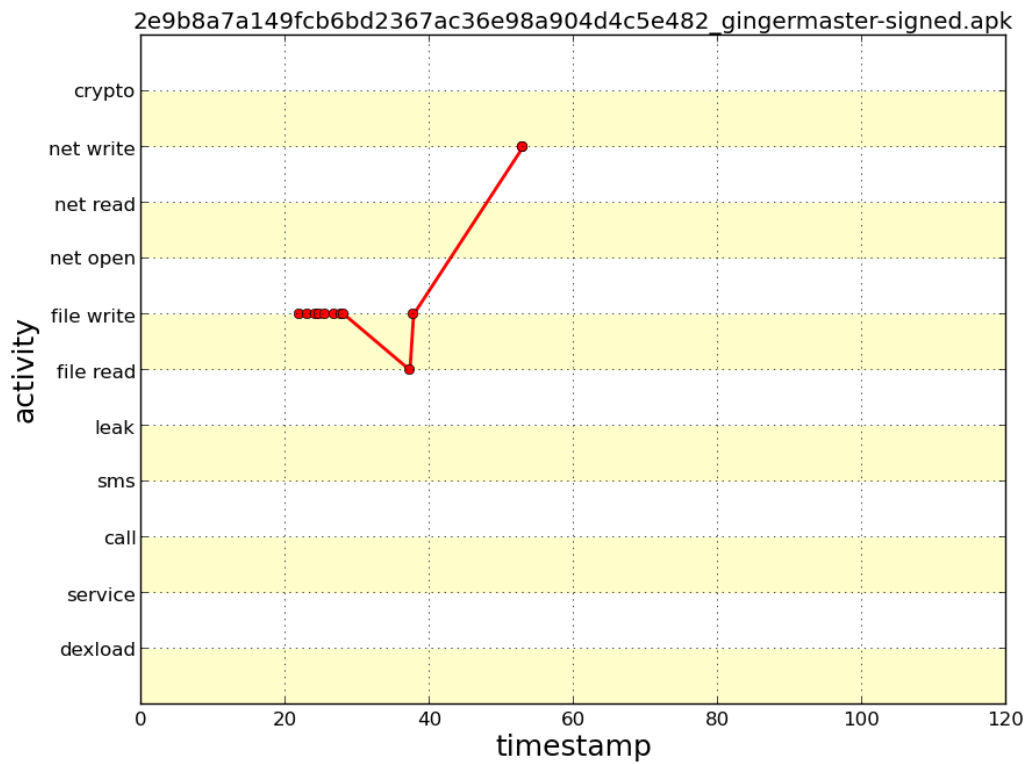


Ilustración 26: TC3-GingerMaster. Resultado final: 1.

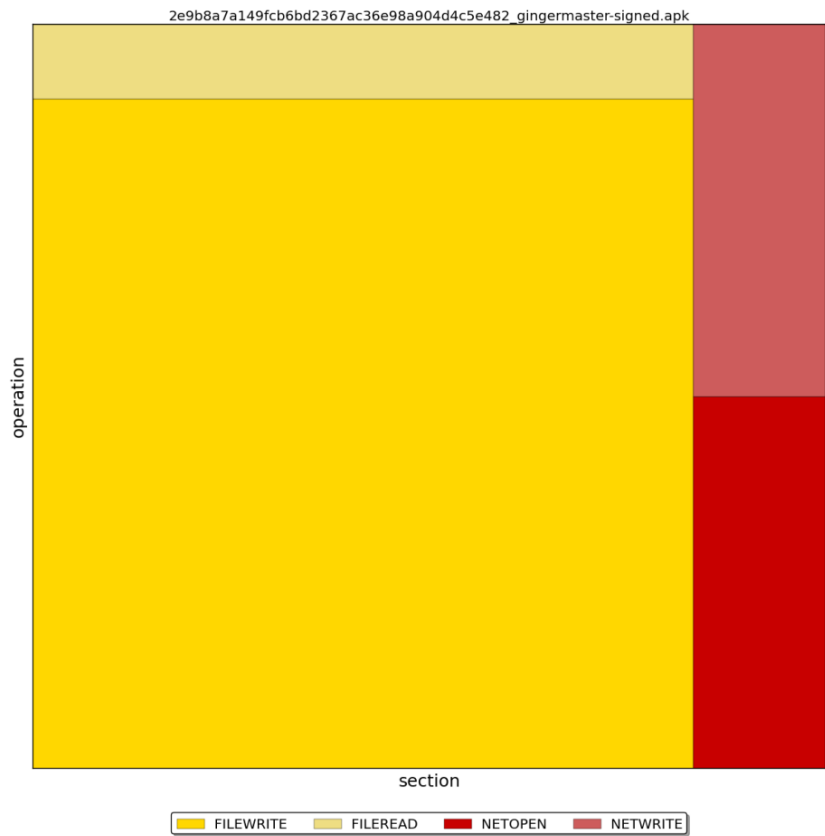


Ilustración 27: TC3-GingerMaster. Resultado final: 2.

Comparando con los resultados obtenidos en la ejecución inicial, no se ha producido filtración de información. También se contempla una disminución en el número de operaciones realizadas. Esto es debido a que las “imágenes” modificadas escondían realmente código malicioso en su interior, que al ser modificado por bytes aleatorios, su funcionalidad maliciosa queda inutilizada.

2.5.4 CASO DE PRUEBA 4: DROIDKUNGFU

Para este caso de prueba se va a volver a emplear una aplicación de la familia *DroidKungFu*. Este tipo de malware descifra en tiempo de ejecución un *exploit*, el cual se encuentra en la carpeta *assets*, dentro del fichero denominado *ratc*. Este *exploit* permite al malware escalar privilegios y poder filtrar información del dispositivo móvil infectado (34).

El objetivo de este caso de prueba se basa en modificar el contenido de este fichero, sustituyendo cada byte por un byte aleatorio, y estudiar su comportamiento.

ANÁLISIS DE COMPORTAMIENTO INICIAL

En esta prueba inicial se ha llevado a cabo una ejecución de la aplicación durante 120 segundos, sin realizar ninguna modificación, utilizando la herramienta *DroidBox*. Los resultados obtenidos se aprecian en las Ilustraciones 28 y 29:

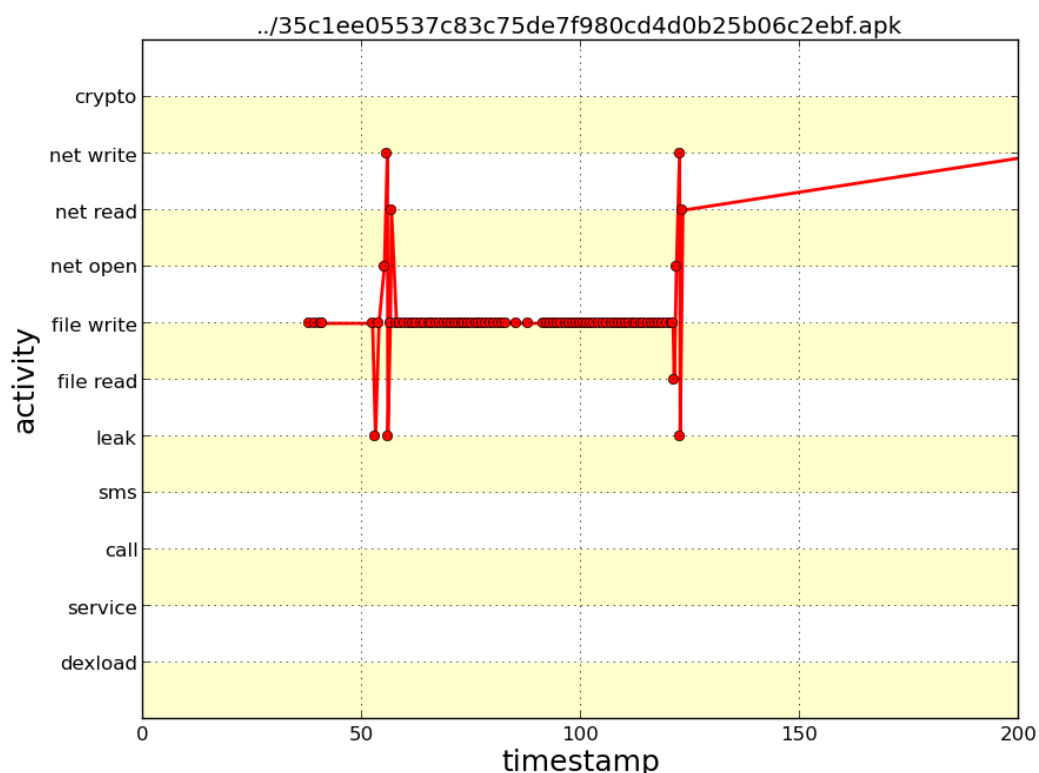


Ilustración 28: TC4-DroidKungFu. Comportamiento ordinario: 1.

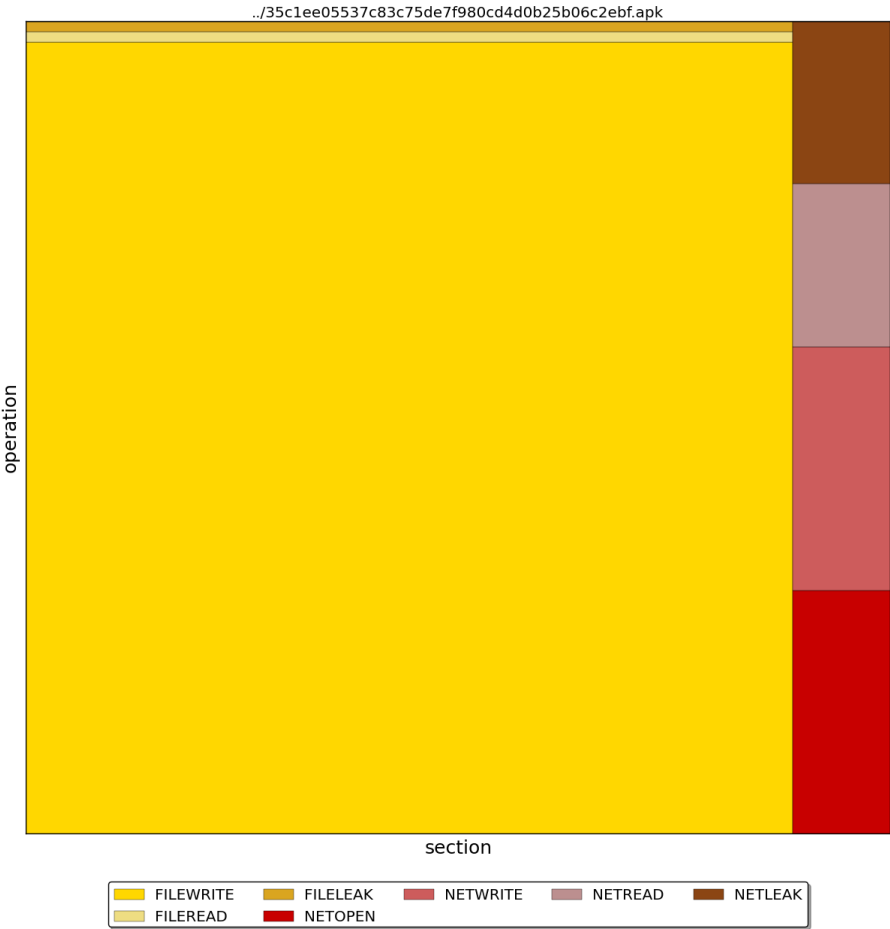


Ilustración 29: TC4-DroidKungFu. Comportamiento ordinario: 2.

De las operaciones producidas durante la ejecución destaca la actividad relacionada con la transmisión de información a través de la red, produciéndose apertura de redes así como lecturas y escrituras a través de la red. También se ha producido filtrado de información, numerosas operaciones de escritura en disco y alguna de lectura.

CASO DE PRUEBA

El caso de prueba realizado consiste en:

Introducción	
Identificador	TC4-DroidKungFu
Nombre	Ofuscación <i>exploit assets</i>
Aplicación analizada	Archivo: 35c1ee05537c83c75de7f980cd4d0b25b06c2ebf.apk MD5: 2358a97d0f9b0e4b7d8e5a8386105c97 SHA1: 35c1ee05537c83c75de7f980cd4d0b25b06c2ebf
Propósito	El fin de este caso de prueba es estudiar el comportamiento tras modificar ficheros que contienen <i>exploits</i> que son descifrados en

	tiempo de ejecución.
Actividades	
Ambiente de prueba	<ul style="list-style-type: none"> • Sistema Santoku-Linux. • AVD con versión Android 2.1. • DroidBox instalado. • ApkTool instalado. • Android SDK instalado. • Dex2Jar instalado. • Conexión a Internet.
Acciones	<ol style="list-style-type: none"> 1. Desensamblado de la aplicación DroidKungFu mediante el uso de APKTool. 2. Identificación de código ofuscado empleando un editor de texto. Para este caso, el fichero <i>ratc</i> presente en la carpeta <i>assets</i>. 3. Modificación del contenido del fichero <i>ratc</i> por bytes aleatorios utilizando un editor de texto. 4. Re-empaquetamiento de la aplicación modificada utilizando ApkTool. 5. Firmar aplicación modificada utilizando Dex2Jar. 6. Ejecución de la aplicación en un sistema AVD versión 2.1 utilizando “Droidbox”. 7. Interacción durante 120 segundos con la UI de la aplicación.
Resultados	
Salida esperada	Comportamiento distinto al de la ejecución inicial.
Salida obtenida	Comportamiento similar al de la prueba inicial.

Tabla 101: TC4-DroidKungFu. Ofuscación *exploit assets*.

Los resultados obtenidos se aprecian en las Ilustraciones 30 y 31:

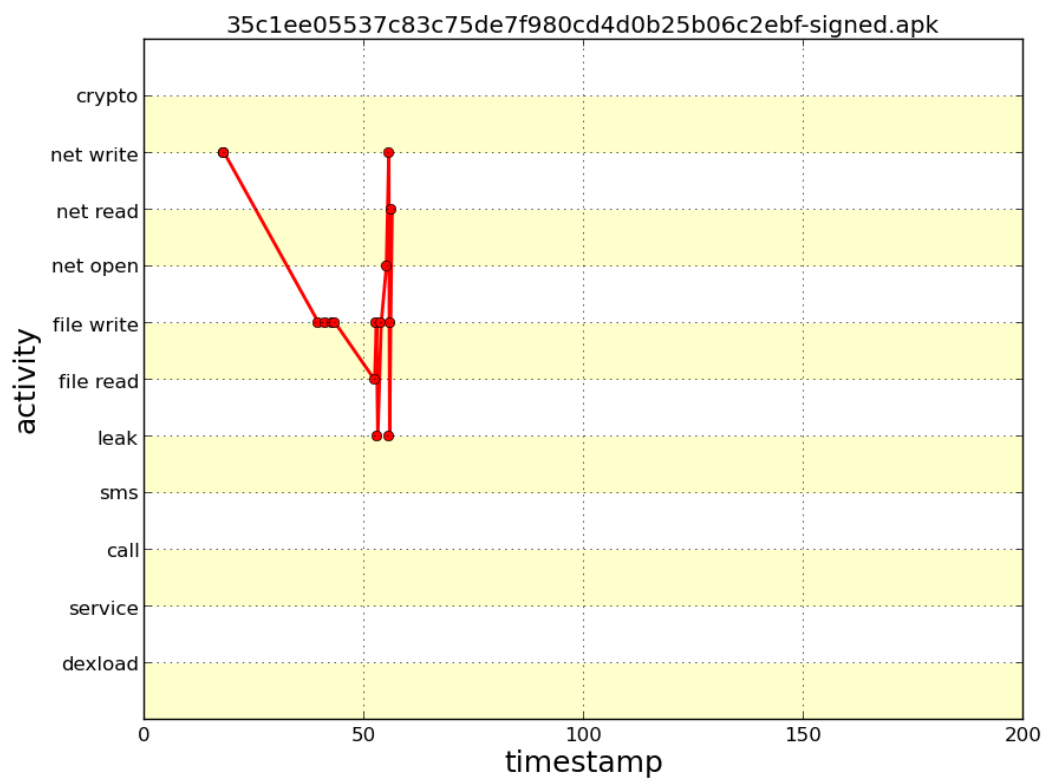


Ilustración 30: TC4-DroidKungFu. Resultado final: 1.

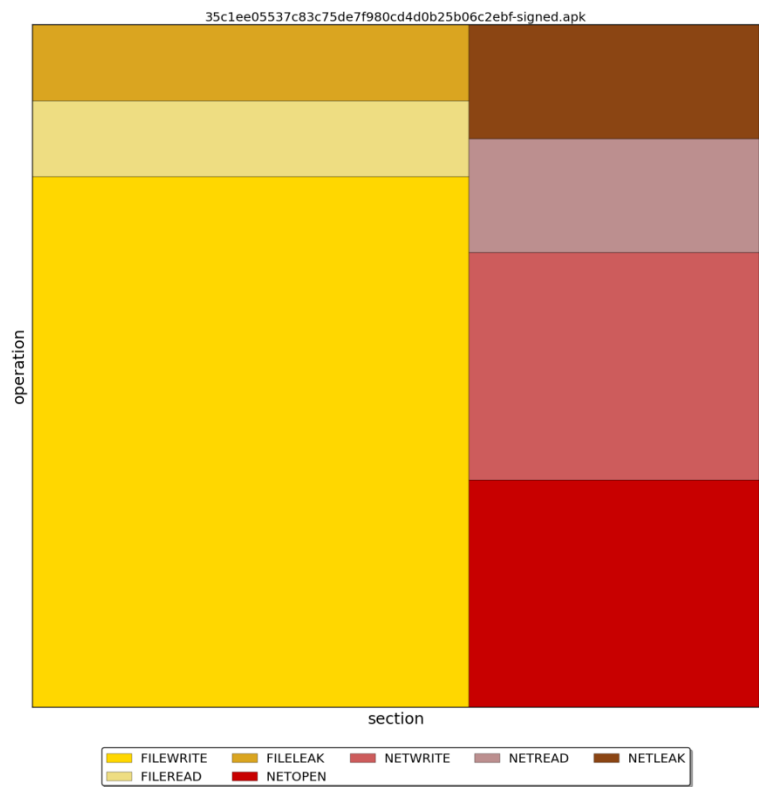


Ilustración 31: TC4-DroidKungFu. Resultado final: 2.

En esta ocasión, se han producido resultados similares al de la prueba inicial. La modificación del *exploit* no ha sido suficiente para detener la funcionalidad del código malicioso, ya que se han producido filtraciones de información y operaciones relacionadas con el recibo y envío de información a través de la red. También se han producido operaciones de escritura y lectura de ficheros.

2.5.5 CASO DE PRUEBA 5: PJAPPS

En este caso de prueba, el objeto de estudio se va a centrar en analizar una aplicación de la familia *Pjapps*. Este tipo de malware se caracteriza por ocultar las direcciones de servidores de comando y control utilizando técnicas de cifrado (1).

El objetivo de este caso de prueba se basa en identificar y modificar la cadena ofuscada que esconde la dirección del servidor C&C. En la aplicación analizada, una de estas cadenas ofuscadas se encuentra en la clase *MainService*. La cadena de texto identificada ha sido: "alfo3gsa3nfsrfo3isd21d8a8fccosm".

ANÁLISIS DE COMPORTAMIENTO ORDINARIO

En esta prueba inicial se ha llevado a cabo una ejecución de la aplicación durante 120 segundos, sin realizar ninguna modificación, utilizando la herramienta *DroidBox*. Los resultados obtenidos se aprecian en las Ilustraciones 32 y 33:

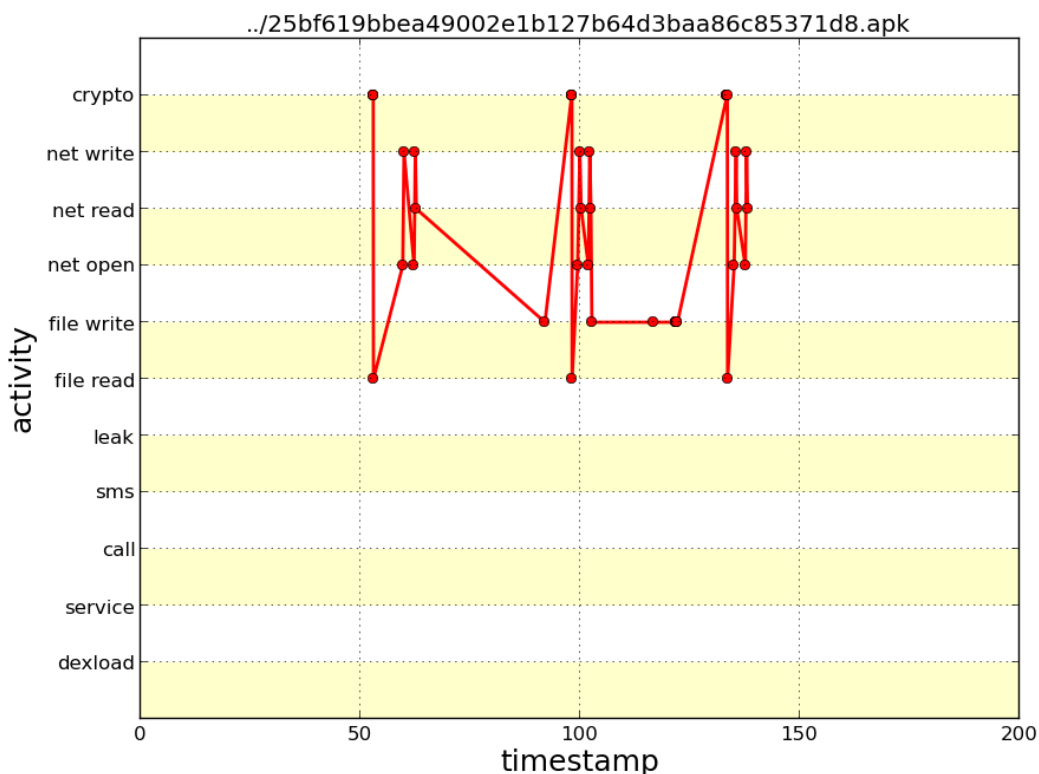


Ilustración 32: TC5-Pjapps. Comportamiento ordinario: 1.

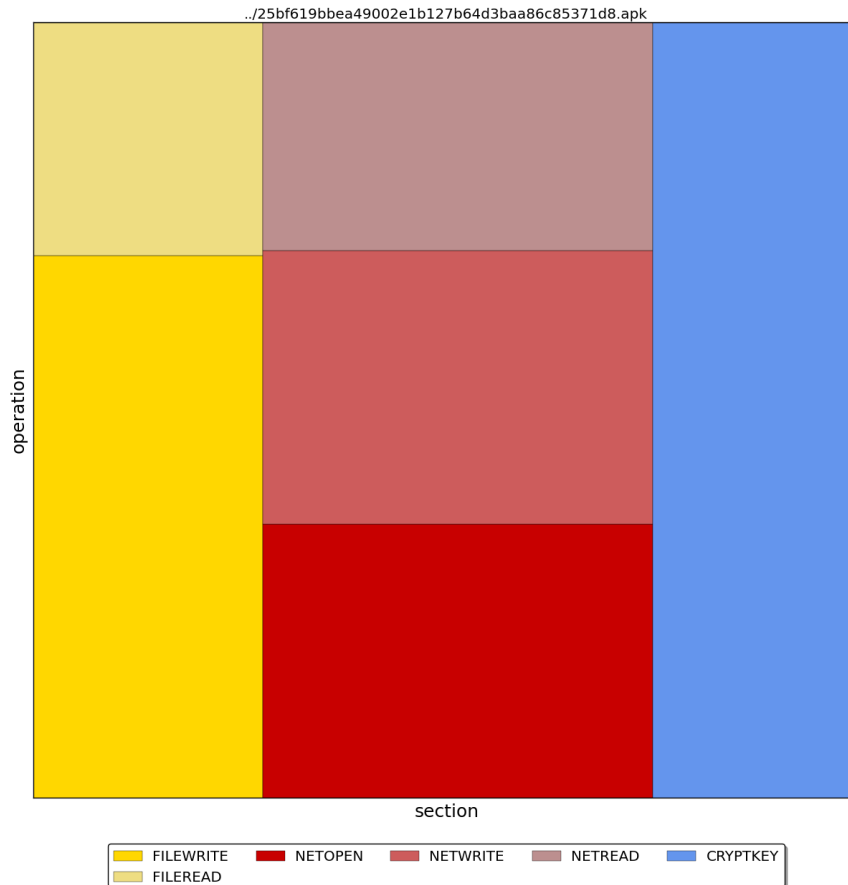


Ilustración 33: TC5-Pjapps. Comportamiento ordinario: 2.

Los resultados obtenidos muestran que se han producido una serie de operaciones criptográficas, que corresponden con el descifrado de las direcciones de los servidores C&C. A su vez, se han producido numerosas operaciones de transmisión y recepción de información a través de la red, que representa la comunicación entre el servidor C&C y el dispositivo móvil. También se han producido operaciones de lectura de ficheros.

En la Ilustración 34 se muestra las direcciones con que ha mantenido una conexión la aplicación analizada:


```

santoku@virtual-machine: ~/Desktop/DroidBox23_Analisis
File Edit Tabs Help
santoku@sa... x santoku@sa... x
, 50, 66, 52, 52, 67, 68, 67, 49, 54, 66, 48, 50, 54, 51, 52, 50, 70, 49, 56, 56
, 56, 48, 54} Algorithm: HmacSHA1
[Network activity]
-----
[Opened connections]
-----
[59.6347429752] Destination: ec2-54-235-102-48.compute-
1.amazonaws.com Port: 80
[62.2067818642] Destination: androidsdk.ads.mp.mydas.mo
bi Port: 80
[99.4258379936] Destination: ec2-107-22-189-105.compute
-1.amazonaws.com Port: 80
[101.872948885] Destination: androidsdk.ads.mp.mydas.mo
bi Port: 80
[135.039083958] Destination: ec2-107-22-189-105.compute
-1.amazonaws.com Port: 80
[137.580047846] Destination: androidsdk.ads.mp.mydas.mo
bi Port: 80

```

Ilustración 34: TC5-Pjapps. Conexiones abiertas iniciales.

Las direcciones a las que se han realizado conexiones son las siguientes:

1. ec2-54-235-102-48.compute-1.amazonaws.com
2. Androidsdk.ads.mp.mydas.mobi
3. ec2-107-22-189-105.compute-1.amazonaws.com

CASO DE PRUEBA

El caso de prueba consiste en lo siguiente:

Introducción	
Identificador	TC5-Pjapps
Nombre	Dirección servidor C&C ofuscada
Aplicación analizada	Archivo: 25bf619bbea49002e1b127b64d3baa86c85371d8.apk MD5: 9a065acda242887ea1d384dca8e1f79e SHA1: 25bf619bbea49002e1b127b64d3baa86c85371d8
Propósito	El fin de este caso de prueba es estudiar el comportamiento tras modificar una variable cifrada que se corresponde con una dirección a un servidor C&C.
Actividades	
Ambiente de prueba	<ul style="list-style-type: none"> • Sistema Santoku-Linux. • AVD con versión Android 2.1. • DroidBox instalado. • ApkTool instalado. • Android SDK instalado. • Dex2Jar instalado.

Acciones	<ul style="list-style-type: none"> • Conexión a Internet. <ol style="list-style-type: none"> 1. Desensamblado de la aplicación Pjapps mediante el uso de APKTool. 2. Identificación de código ofuscado empleando un editor de texto. La cadena “alfo3gsa3nfdsrfo3isd21d8a8fccosm” presente en la clase MainService. 3. Modificación de la cadena ofuscada por una cadena de mismo tamaño formada por bytes aleatorios mediante un editor de texto. 4. Re-empaquetamiento de la aplicación modificada utilizando ApkTool. 5. Firmar aplicación modificada utilizando Dex2Jar. 6. Ejecución de la aplicación en un sistema AVD versión 2.1 utilizando “Droidbox”. 7. Interacción durante 120 segundos con la UI de la aplicación.
Resultados	
Salida esperada	El número de conexiones realizadas debe ser menor al de la prueba original.
Salida obtenida	No se ha realizado conexión a la dirección: ec2-54-235-102-48.compute-1.amazonaws.com

Tabla 102: TC5-Pjapps. Dirección servidor C&C ofuscada.

Los resultados obtenidos tras realizar el caso de uso se muestra en las Ilustraciones 35 y 36:

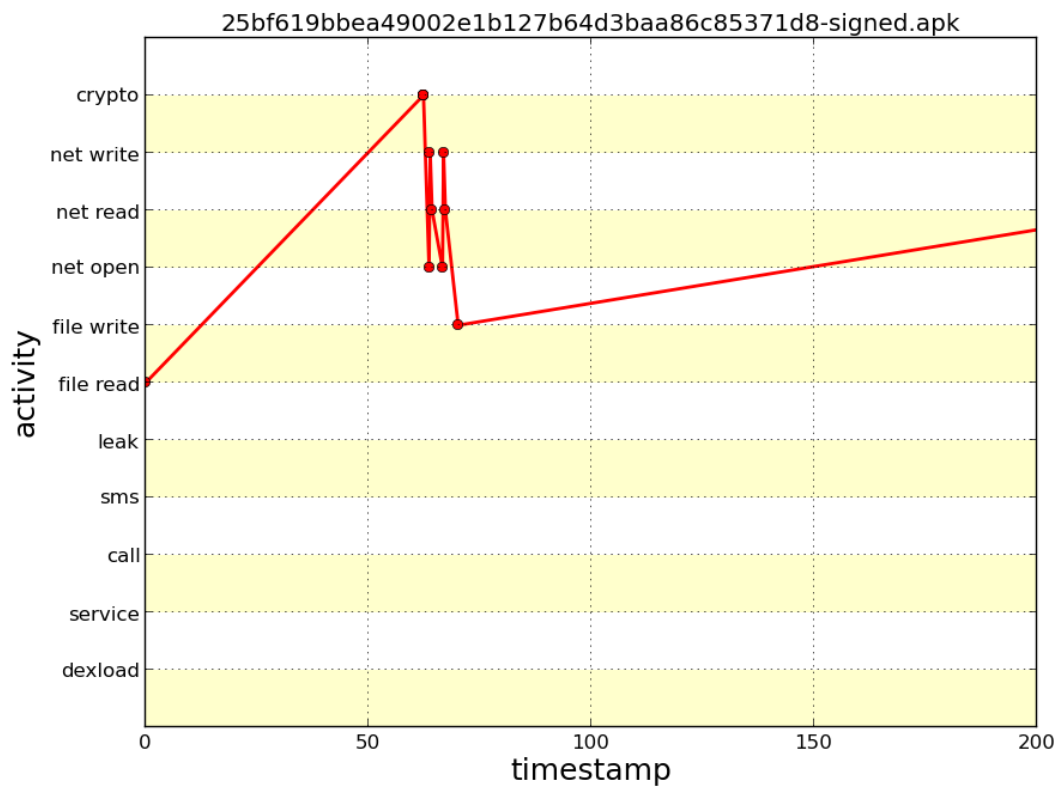


Ilustración 35: TC5-Pjapps. Resultado final: 1.

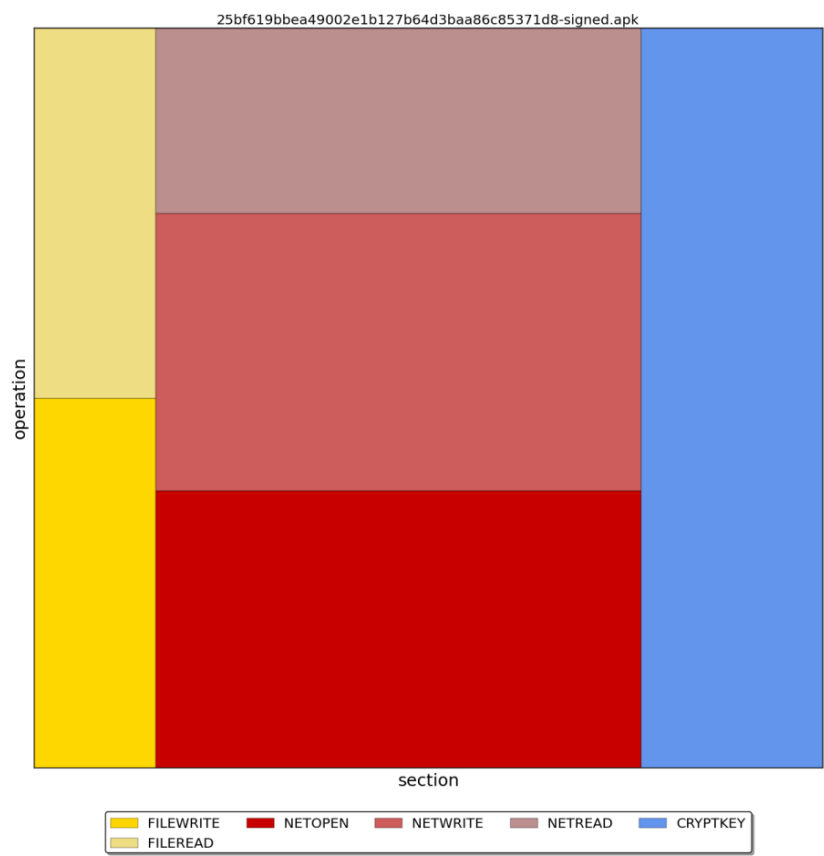
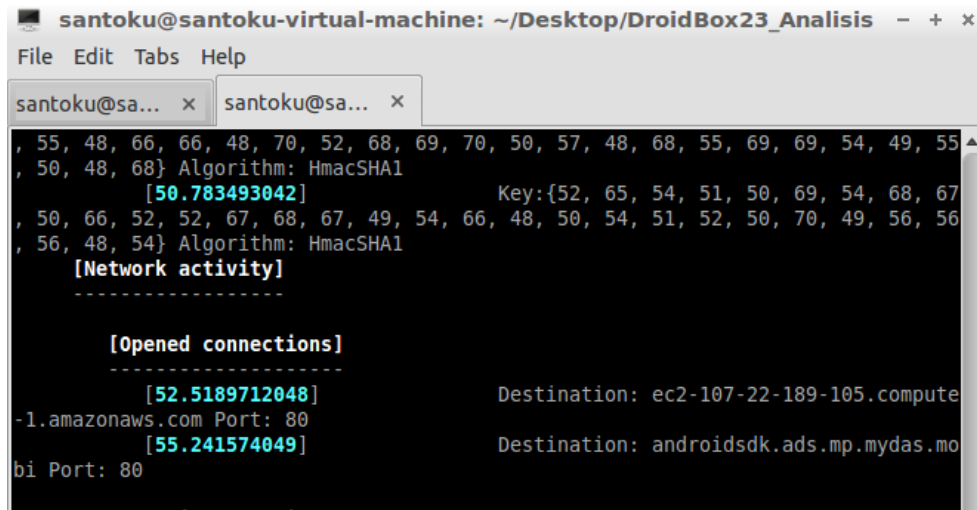


Ilustración 36: TC5-Pjapps. Resultado final: 2.

Las operaciones producidas durante la ejecución de la aplicación modificada son similares, conexiones y transmisión de datos a través de la red acompañada de operaciones criptográficas. Cabe destacar la disminución en el número de operaciones total. En la siguiente Ilustración se muestran las conexiones establecidas durante la ejecución:



```
santoku@santoku-virtual-machine: ~/Desktop/DroidBox23_Analisis
File Edit Tabs Help
santoku@sa... x santoku@sa... x
, 55, 48, 66, 66, 48, 70, 52, 68, 69, 70, 50, 57, 48, 68, 55, 69, 69, 54, 49, 55
, 50, 48, 68} Algorithm: HmacSHA1
[50.783493042] Key:{52, 65, 54, 51, 50, 69, 54, 68, 67
, 50, 66, 52, 52, 67, 68, 67, 49, 54, 66, 48, 50, 54, 51, 52, 50, 70, 49, 56, 56
, 56, 48, 54} Algorithm: HmacSHA1
[Network activity]
-----
[Opened connections]
-----
[52.5189712048] Destination: ec2-107-22-189-105.compute
-1.amazonaws.com Port: 80
[55.241574049] Destination: androidsdk.ads.mp.mydas.mo
bi Port: 80
```

Ilustración 37: TC5-Pjapps. Conexiones abiertas finales.

En este caso sólo se han producido dos conexiones a servidores distintos:

1. Androidsdk.ads.mp.mydas.mobi
2. ec2-107-22-189-105.compute-1.amazonaws.com

La conexión al servidor ec2-54-235-102-48.compute-1.amazonaws.com no se ha producido en este caso de prueba, por lo que todo indica que tras la cadena ofuscada modificada se escondía dicha dirección.

CAPÍTULO 3

DISEÑO

En este capítulo se presentará el diseño del sistema a desarrollar. Se definirá la arquitectura del sistema y los componentes que la forman. También se detallará el diagrama de clases del sistema, presentando las clases que lo componen y las relaciones entre ellos.

3.1 ARQUITECTURA DEL SISTEMA

En este apartado se expondrá el diagrama de componentes que representa la arquitectura del sistema diseñada y una explicación de cada uno de los componentes que lo forman.

En la Ilustración 38 se muestra el diseño de componentes del sistema a desarrollar:

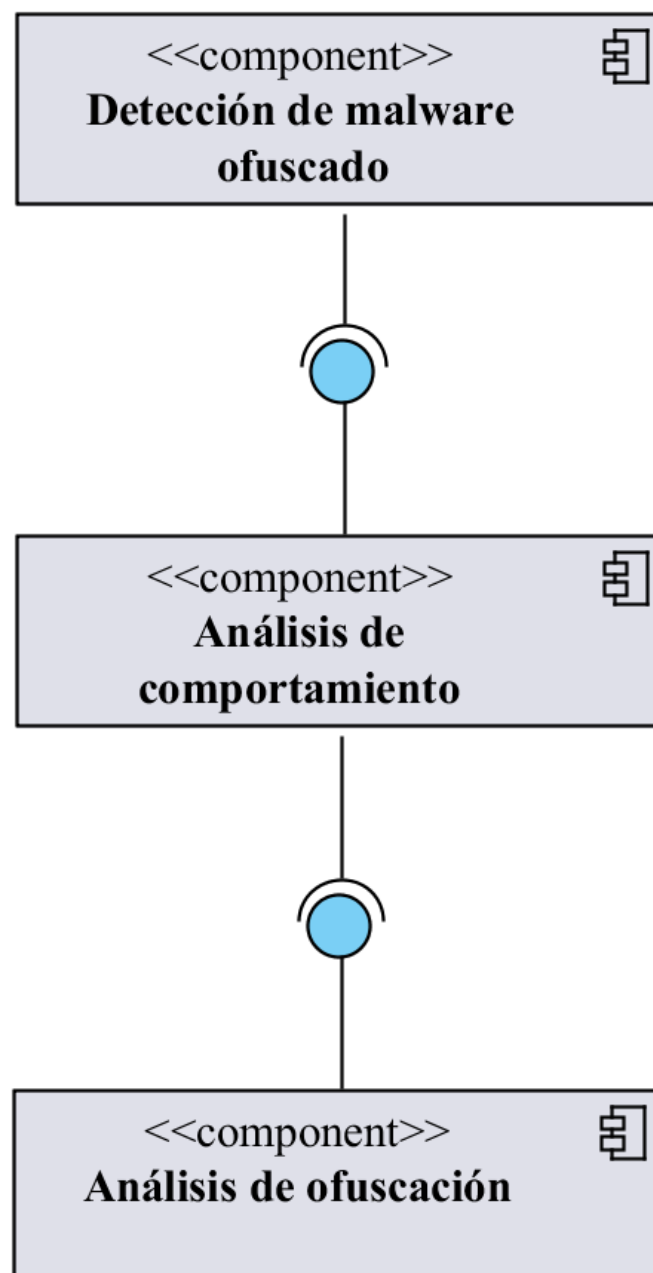


Ilustración 38: Arquitectura del sistema.

Se presentan tres componentes principales:

- **Análisis de ofuscación:** este componente se centra en estudiar los posibles artefactos ofuscados presentes en la aplicación a analizar.
- **Análisis de comportamiento:** su objetivo consiste en estudiar la conducta de la aplicación en ejecución.
- **Detección de malware ofuscado:** la finalidad de este componente se focaliza en nutrirse de la información generada por los dos integrantes anteriores del sistema para descubrir manifestaciones de *malware* ofuscado.

3.1.1 ANÁLISIS DE OFUSCACIÓN

El objetivo de este componente se centra en el estudio automático de artefactos y elementos ofuscados presentes en la aplicación a analizar.

Las funcionalidades que debe desempeñar este componente son las siguientes:

- **Descompilador:** su tarea consiste en traducir código de bajo nivel de abstracción a un lenguaje de mayor nivel de abstracción. En concreto, en este sistema el descompilador debe estar capacitado para traducir el código ejecutable de aplicaciones Android a su código fuente, en algún lenguaje de alto o medio nivel de abstracción.
- **Analizador estático:** se centra en el estudio del código fuente obtenido tras la descompilación. Este estudio se centra en realizar un análisis estático, obteniendo información de la estructura que presenta la aplicación, y estudiando los distintos elementos que forman la aplicación.

Con los resultados obtenidos tras la realización de la fase de análisis, se han determinado los siguientes tipos de análisis estáticos a realizar por el sistema:

- Descubrimiento de cadenas ofuscadas: este proceso consiste en identificar las posibles cadenas de texto ofuscadas presentes en el código fuente de la aplicación.
- Estudio de ficheros *assets*: muchas aplicaciones Android presentan en su estructura una carpeta denominada *assets*, utilizada por los desarrolladores para almacenar recursos de distinto tipo. Los atacantes aprovechan la existencia de este directorio para almacenar ficheros que esconden códigos maliciosos capaces de explotar vulnerabilidades presentes en la aplicación.
- Condicionales: este análisis se centra en identificar todas las sentencias condicionales presentes en el código fuente. Su finalidad es poder estudiar, a posteriori, todos los posibles flujos de ejecución dentro de la aplicación.

- Librerías criptográficas: otra posible fuente de ofuscación se encuentra en las llamadas a APIs criptográficas, para cifrar o descifrar elementos que son utilizados por el código malicioso presente en las aplicaciones. Con este análisis se identifica los fragmentos del código fuente que hacen uso de este tipo de llamadas.
- **Test Case Maker:** es el motor de mutación de artefactos ofuscados del sistema. Tras realizar el análisis estático de la aplicación, se realizarán una serie de modificaciones que afecten a los elementos analizados estáticamente, generando un caso de prueba por cada modificación. Con estas mutaciones se pretende conseguir estudiar la variabilidad en el comportamiento de las aplicaciones Android a analizar.
- **Repackaging:** este integrante del sistema se centra en recompilar la aplicación con las modificaciones realizadas sobre el código fuente.

3.1.2 ANÁLISIS DE COMPORTAMIENTO

Este componente se centra en estudiar y registrar el comportamiento de la aplicación en tiempo de ejecución. Este componente está formado por dos integrantes bien diferenciados:

- **Analizador dinámico:** su objetivo es llevar a cabo un análisis dinámico de la aplicación Android, estudiando el comportamiento y registrando los eventos sucedidos en tiempo de ejecución.
- **Tester automático:** este integrante se encarga de apoyar al analizador automático en su función. Se focaliza en la realización de un análisis de la interfaz de usuario que presenta la aplicación. De esta manera, este mecanismo permite interactuar con todos los artefactos presentes en la interfaz de usuario de manera automática y adaptándose a cada aplicación. Con esto se consigue un análisis dinámico personalizado para cada aplicación.

3.1.3 DETECCIÓN DE MALWARE OFUSCADO

Este componente es el encargado de analizar la información recogida por los dos componentes anteriores, Análisis de ofuscación y Análisis de comportamiento. Se centra en organizar y estudiar todos estos datos con el fin de detectar la presencia de código malicioso de carácter ofuscado.

En primer lugar, este módulo se alimenta de la información recogida en las ejecuciones ordinarias, aquellas en que la aplicación no haya sufrido ningún tipo de modificación.

En segundo lugar se calcula un ratio de error tolerable, que indica la desviación aceptada entre ejecuciones ordinarias.

Con este ratio se establece un umbral, el cual toma como centro la media de los resultados ordinarios. El límite superior e inferior se establece utilizando el ratio de error tolerable calculado respecto a la media, de manera positiva y negativa, respectivamente. Todos los resultados obtenidos son filtrados y analizados frente a este umbral definido. Si los resultados se encuentran dentro de este umbral, se considera que el resultado se asemeja al comportamiento ordinario de la aplicación, pero si por el contrario se encuentra fuera de este umbral, el resultado es considerado como comportamiento anómalo.

Este proceso se realiza para cada uno de los tipos de operaciones analizados durante la ejecución. Finalmente, se compara el resultado de cada uno de los tipos de operaciones, y si se han producido un porcentaje mayor a la media de comportamientos anómalos respecto al comportamiento ordinario, se considera que dicho caso de prueba ha afectado al comportamiento del *malware* ofuscado.

En la Ilustración 39 se resume todo lo comentado en los párrafos anteriores:

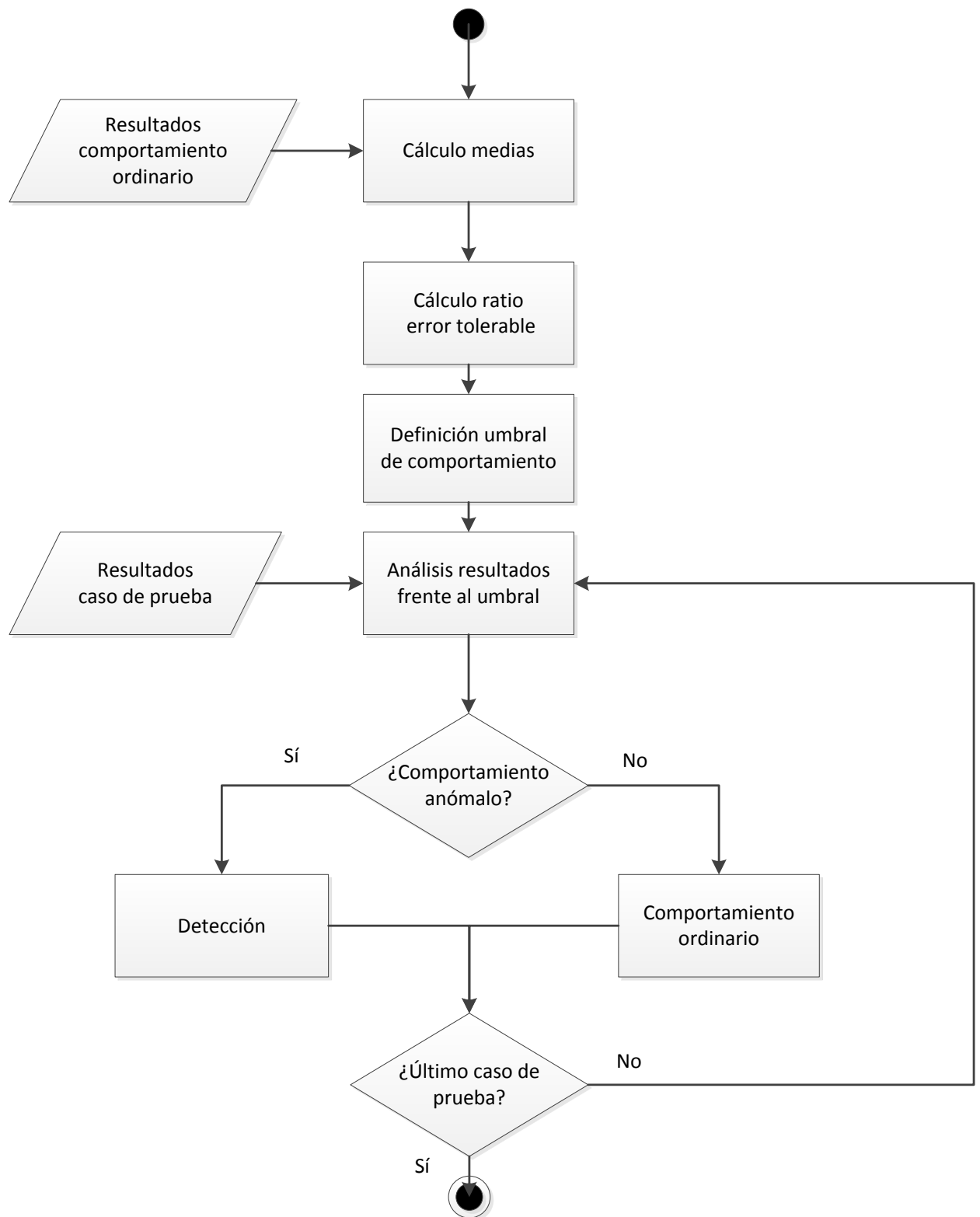


Ilustración 39: Diagrama de flujo: Módulo Detección.

3.2 DIAGRAMA DE CLASES

En este diagrama de clases (Ilustración 40) se presentan las principales clases del sistema y sus relaciones:

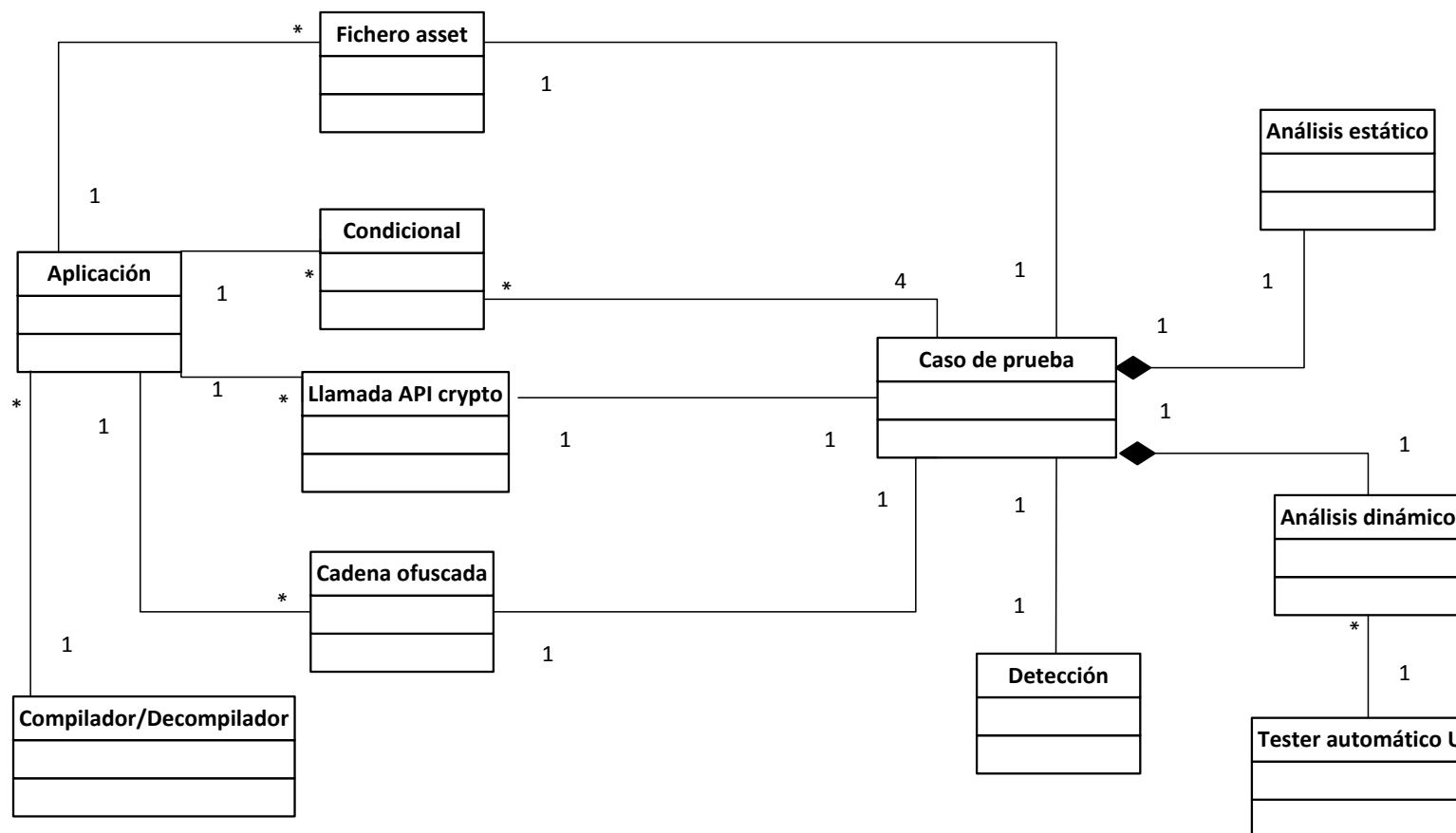


Ilustración 40: Diagrama de clases

Para apreciar mejor las relaciones entre clases, se va a dividir el diagrama anterior en tres partes, cada una de las cuales se corresponde con un componente definido en la arquitectura.

3.2.1 CLASES: ANÁLISIS DE OFUSCACIÓN

En la primera agrupación, correspondiente a la Ilustración 41, se aprecian las clases que forman el componente Análisis de ofuscación.

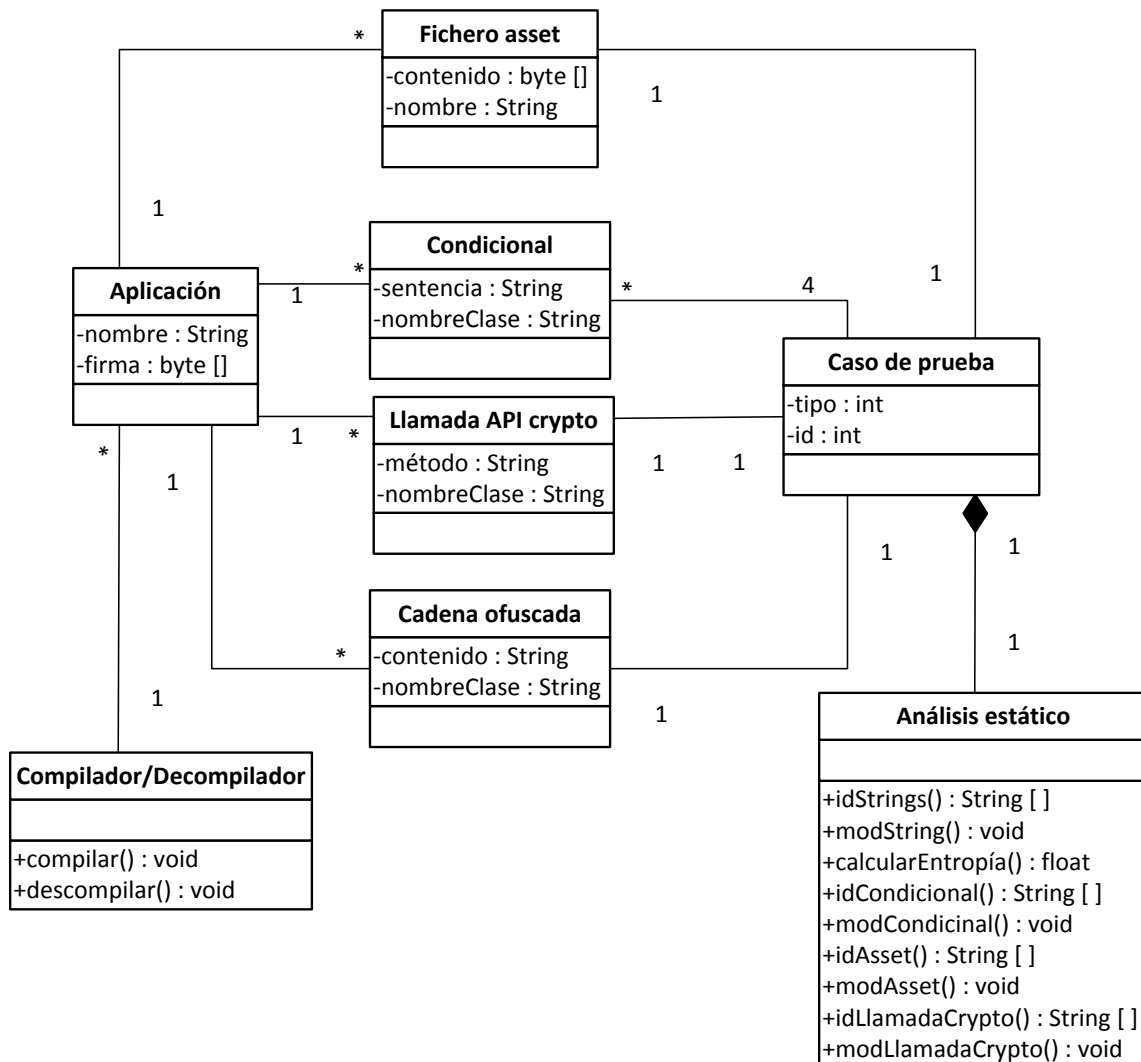


Ilustración 41: Clases: Análisis de ofuscación.

Se detalla que una aplicación, haciendo referencia a la *app* Android que se quiere analizar, puede incluir de cero a muchos ficheros en la carpeta *assets*. También puede presentar de cero a muchas sentencias condicionales en el código fuente, de cero a muchas llamadas a APIs criptográficas y de cero a muchas cadenas ofuscadas. Todos estos elementos son los posibles artefactos utilizados para realizar el estudio de presencia de *malware* ofuscado en la aplicación. También existe la entidad

Compilador/Decompilador, el encargado de utilizar la técnica de *repackaging* con la aplicación a analizar, obteniendo el código fuente de la aplicación original y generando el ejecutable de la aplicación modificada. También se define que existe un Caso de prueba por cada fichero *assets* y cuatro casos de prueba distintos en cuanto al análisis de sentencias condicionales en el código fuente. Asimismo, se realiza un caso de prueba por cada cadena de texto ofuscada y un caso de prueba por cada llamada a librerías criptográficas realizadas en el código fuente de la aplicación, mientras que a clase Caso de prueba siempre está compuesta por un Análisis estático.

En las siguientes tablas se detallarán los atributos y operaciones cada una de las clases que integran el componente especificado:

Clase	Fichero <i>asset</i>
Atributos	-contenido: secuencia de bytes que forman el fichero.
	-nombre: cadena de texto con que se denomina dicho fichero.
Operaciones	-

Tabla 103: Clase Fichero *asset*.

Clase	Condicional
Atributos	-sentencia: cadena de texto que forma la sentencia condicional.
	-nombreClase: cadena de texto que identifica a la clase en la que está incluida la sentencia condicional.
Operaciones	-

Tabla 104: Clase Condicional.

Clase	Llamada API <i>crypto</i>
Atributos	-método: es la cadena de texto utilizada para referenciar el método utilizado.
	-nombreClase: cadena de texto con la que se denomina la clase donde se encuentra dicha llamada criptográfica.
Operaciones	-

Tabla 105: Clase Llamada API *crypto*.

Clase	Cadena ofuscada
Atributos	-contenido: cadena de texto ofuscada.
	-nombreClase: cadena de texto con la que se denomina la clase en la que se encuentra la cadena ofuscada.
Operaciones	-

Tabla 106: Clase Cadena ofuscada.

Clase	Caso de prueba
Atributos	-tipo: entero que indica el tipo de caso de prueba realizado.
	-id: entero con el que se identifica unívocamente cada caso de prueba.
Operaciones	-

Tabla 107: Clase Caso de prueba.

Clase	Análisis estático
Atributos	-
Operaciones	+idStrings(): capacidad de encontrar todas las cadenas de texto presentes en el código fuente de la aplicación.
	+modString(): mutación de una cadena ofuscada hallada en el código fuente de la aplicación.
	+calcularEntropía(): cálculo de la entropía de una cadena de texto para conocer su nivel de aleatoriedad.
	+idCondicional(): localización de todas las sentencias condicionales en el código fuente.
	+modCondicional(): mutación de la sentencia condicional identificada.
	+idAsset(): localización de todos los ficheros <i>assets</i> presentes en la estructura de la aplicación.
	+modAsset(): mutación del contenido de un fichero <i>assets</i> .
	+idLlamadaCrypto(): localización de todas las llamadas a librerías criptográficas realizadas en el código fuente de la aplicación.
	+modLlamadaCrypto(): mutación del método que realiza la llamada a APIs criptográficas.

Tabla 108: Clase Análisis estático.

Clase	Compilador/Decompilador
Atributos	-
Operaciones	+compilar(): capacidad de generar el código ejecutable Android a partir de un código fuente.
	+decompilar(): capacidad de obtener el código fuente y recursos a partir de una aplicación Android en formato <i>.apk</i> .

Tabla 109: Clase Compilador/Decompilador.

3.2.2 CLASES: ANÁLISIS DE COMPORTAMIENTO

En una segunda agrupación, representada en la Ilustración 42, agrupa las clases presentes en el componente Análisis de comportamiento. Este diagrama define que un Caso de prueba está siempre compuesto por un Análisis dinámico. A su vez, existe un Tester automático de la interfaz de usuario para todos los posibles Análisis dinámicos de una misma aplicación.

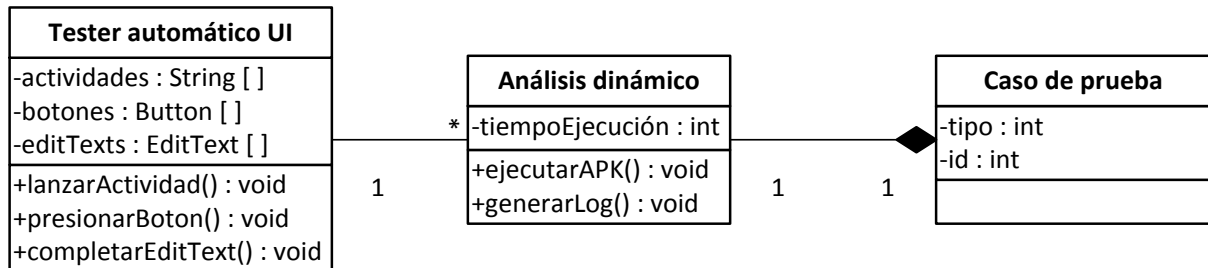


Ilustración 42: Clases: Análisis de comportamiento.

En las siguientes tablas se detallarán los atributos y operaciones cada una de las clases que integran el componente especificado:

Clase	Análisis dinámico
Atributos	-tiempoEjecución: entero que indica el tiempo empleado en realizar el análisis dinámico de la aplicación.
Operaciones	+ejecutarAPK(): capacidad de poner en ejecución la aplicación y registrar su comportamiento. +generarLog(): almacenamiento en un fichero de todos los eventos ocurridos durante el análisis de comportamiento.

Tabla 110: Clase Análisis dinámico.

Clase	Tester automático UI
Atributos	-actividades: conjunto de actividades que presenta la aplicación Android. -botones: conjunto de botones presentes en las interfaces de usuario de una aplicación Android. -editTexts: conjunto de artefactos de interfaz destinados a la introducción de texto por parte del usuario.
Operaciones	+lanzarActividad(): capacidad de ejecutar una actividad de una aplicación Android. +presionarBoton(): debe ser capaz de presionar botones de la interfaz de la aplicación de manera automática. +completarEditText(): introducción automática de texto en la interfaz de usuario de la app.

Tabla 111: Clase Tester automático UI.

3.2.3 CLASES: DETECCIÓN DE MALWARE OFUSCADO

En la última agrupación se detalla las clases que integran al componente Detección de *malware* ofuscado. En la Ilustración 43, se aprecia que por cada Caso de prueba realizada se genera una Detección, la cual recoge toda la información referente al comportamiento de la aplicación durante el test case.

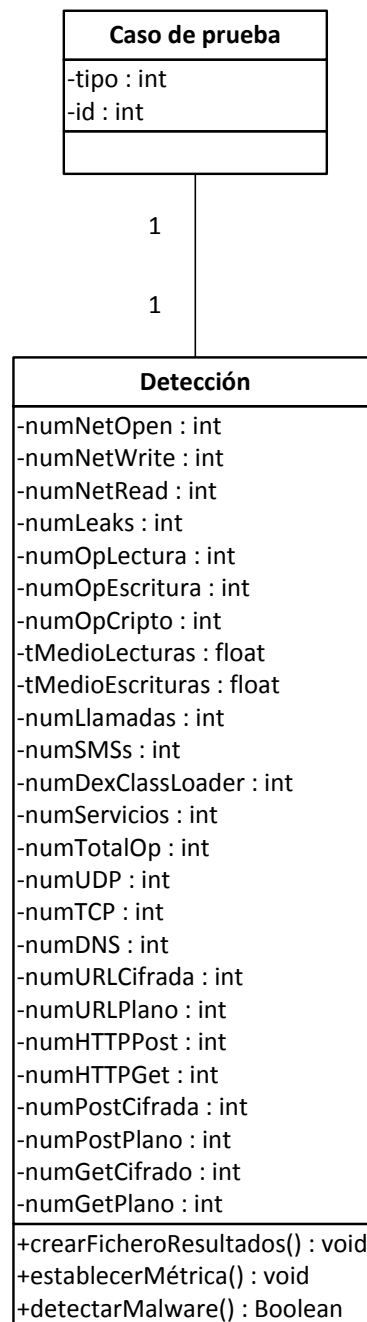


Ilustración 43: Clases: Detección de malware ofuscado.

En las siguientes tablas se detallarán los atributos y operaciones cada una de las clases que integran el componente especificado:

Clase	Detección
Atributos	- numNetOpen : entero que indica la cantidad de conexiones abiertas durante el análisis dinámico.
	- numNetRead : entero que indica la cantidad de conexiones entrantes durante el análisis dinámico.
	- numNetWrite : entero que indica la cantidad de conexiones salientes durante el análisis dinámico.
	- numLeaks : entero que indica la cantidad de operaciones de filtrado de información ocurridas durante el análisis dinámico.
	- numOpLectura : entero que indica el número de lecturas a ficheros han sido realizadas.
	- numOpEscritura : entero que indica el número de escrituras a ficheros han sido realizadas.
	- numOpCripto : entero que indica la cantidad de operaciones relacionadas con actividades de carácter criptográficas realizadas durante el análisis.
	- tMedioLecturas : flotante que indica una aproximación del tiempo invertido en operaciones de lectura en disco durante el análisis.
	- tMedioEscrituras : flotante que expresa un tiempo aproximado empleado en actividades de escritura en disco durante el análisis.
	- numLlamadas : entero que indica el número de llamadas registradas durante el análisis.
	- numSMSs : entero que expresa la cantidad de intentos de envío de SMSs producidos durante el análisis de comportamiento.
	- numServicios : entero que hace referencia al número de servicios iniciados durante la ejecución de la aplicación.
	- numDexClassLoader : entero que hace referencia al número de clases que se han cargado dinámicamente durante la ejecución de la aplicación.
	- numTotalOp : entero que expresa el sumatorio de todas las actividades de distinto tipo que se han producido durante el análisis.
	- numUDP : entero que indica el total de paquetes UDP que han intervenido en el tráfico de red durante la ejecución de la aplicación.
	- numTCP : entero que muestra el total de paquetes TCP que han intervenido en el tráfico de red durante la ejecución de la aplicación.
	- numDNS : entero que expresa el total de conexiones DNS que han intervenido en el tráfico de red durante la ejecución de la aplicación.
	- numURLCifrada : entero que indica el número de direcciones URL cifradas de las peticiones DNS.

	-numURLPlano: entero que indica el número de direcciones URL en claro de las peticiones DNS.
	-numHTTPPost: entero que expresa el número de operaciones HTTP POST producidas durante la ejecución de la aplicación.
	-numHTTPGet: entero que expresa el número de operaciones HTTP GET producidas durante la ejecución de la aplicación.
	-numPostCifrada: entero que expresa el número de operaciones HTTP POST cuya información se encuentra cifrada.
	-numPostPlano: entero que expresa el número de operaciones HTTP POST cuya información se envía en claro durante la ejecución de la aplicación.
	-numGetCifrada: entero que expresa el número de operaciones HTTP GET cuya información se encuentra cifrada.
	-numGetPlano: entero que expresa el número de operaciones HTTP GET cuya información se envía en claro durante la ejecución de la aplicación.
Operaciones	+crearFicheroResultados(): capacidad de creación un fichero en que se reflejen los resultados de detección de <i>malware</i> tras realizar el caso de prueba.
	+establecerMétrica(): capacidad para establecer un umbral utilizado para decidir sobre la presencia de código malicioso ofuscado.
	+detectarMalware(): capacidad de estudiar los resultados obtenidos con el umbral establecido para reconocer si cierta ejecución ha modificado su comportamiento malicioso.

Tabla 112: Clase Detección.

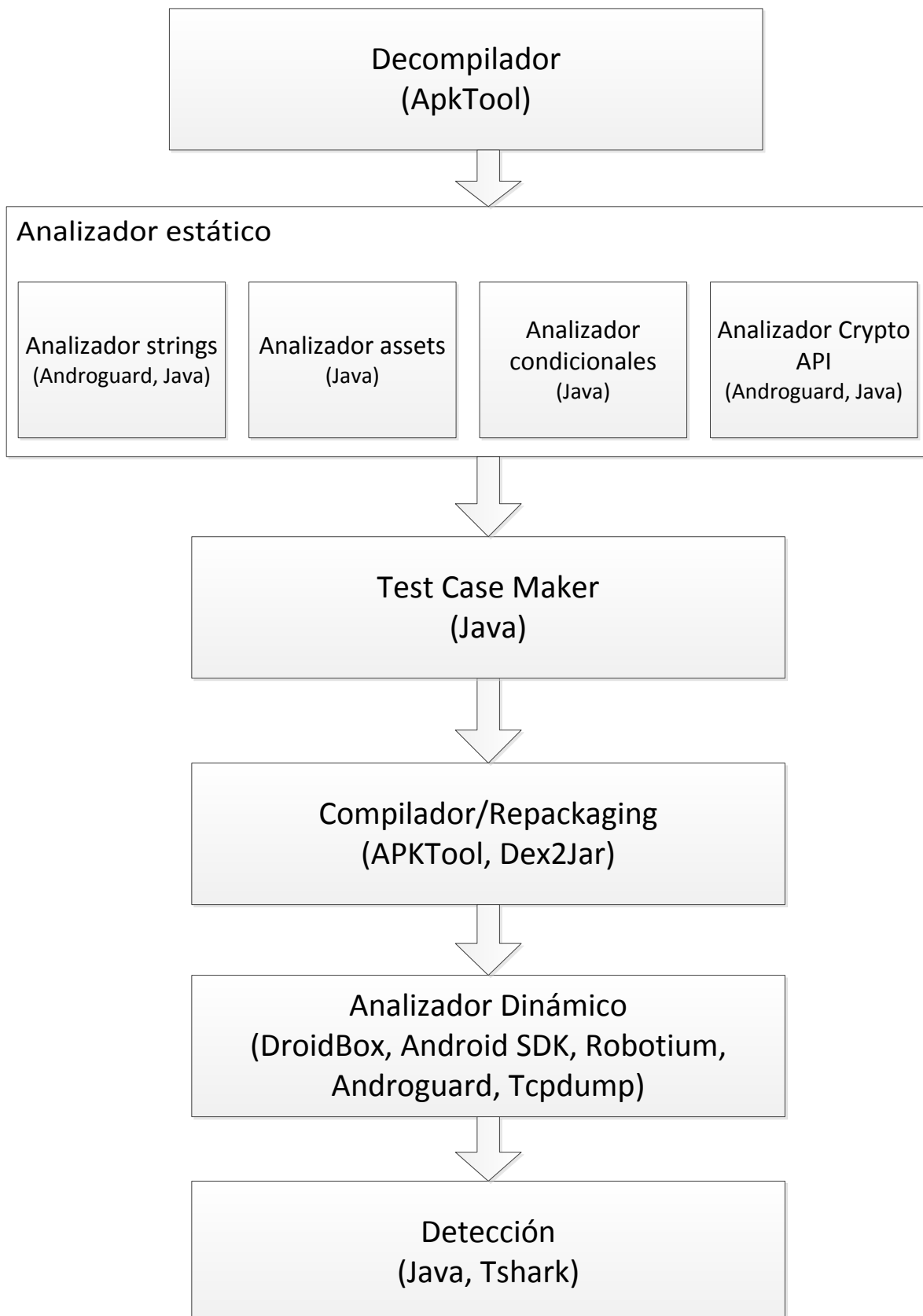
CAPÍTULO 4

IMPLEMENTACIÓN

En este capítulo se explican las decisiones tomadas durante la etapa de implementación del TFG, tras haber realizado el análisis y diseño del sistema. Se presentaran las herramientas externas utilizadas y su integración con el sistema, así como elementos claves de la codificación del programa.

En la Ilustración 44 se muestra una visión global del flujo de ejecución del sistema, de los principales integrantes y de las tecnologías empleadas para su implementación. En el diseño de la arquitectura del sistema realizado y expuesto en el capítulo anterior, se han definido tres componentes principales: Análisis de ofuscación, Análisis de comportamiento y Detección de malware ofuscado. Cada uno de los elementos destacados en la Ilustración 44, se encuentra integrado en alguno de estos tres componentes.

A continuación se expondrá una descripción detallada de la tecnología utilizada en cada una de las partes que conforman el sistema, destacando las soluciones de implementación de mayor notoriedad.

**Ilustración 44: Integrantes del sistema.**

4.1 DECOMPILADOR

Este elemento del sistema tiene como objetivo obtener el código fuente de una aplicación Android a partir de su código ejecutable.

Esta parte del sistema ha sido desarrollada utilizando la herramienta *ApkTool*. Esta herramienta diseñada para la ingeniería inversa, permite obtener una fuerte aproximación del código fuente a partir de una aplicación Android en formato *.apk*. El código fuente que devuelve *ApkTool* se encuentra en formato *smali*. La herramienta *ApkTool* también es capaz de recuperar ficheros utilizados como recursos de la aplicación, como son los ficheros presentes en directorios *res* o *assets*.

Con la aplicación descompilada devuelta por *ApkTool* se obtienen los elementos necesarios para que el sistema empiece a trabajar.

4.2 ANALIZADOR ESTÁTICO

Este integrante del sistema, presente en el componente Análisis de ofuscación de la arquitectura, se centra en analizar el código obtenido tras la descompilación de la aplicación Android, con el fin de identificar artefactos ofuscados.

El sistema definido realiza cuatro tipos principales de análisis estático del código fuente de la aplicación:

- 4.2.1 ANALIZADOR DE CADENAS DE TEXTO

El objetivo de este elemento del sistema es identificar todas las posibles cadenas ofuscadas presentes en el código fuente. Como se explicó en el capítulo de Análisis, muchos tipos de *malware* esconden componentes maliciosos tras cadenas de texto ofuscadas en el código de la aplicación, como pueden ser direcciones a servidores de comando y control.

Las tecnologías utilizadas para la implementación de esta parte del sistema se han basado en la herramienta *Androguard* combinada con Java.

La aplicación *Androguard*, escrita en lenguaje Python, permite realizar distintos tipos de análisis, mediante ingeniería inversa, de una aplicación Android. Para este caso en concreto, se ha implementado un módulo para *Androguard* utilizando la API que proporciona dicha herramienta. En este módulo, a partir de la aplicación Android en formato *.apk*, se obtiene el fichero *classes.dex*, a partir del cual se genera un fichero que contiene todos los strings presentes en el código fuente de la aplicación.

Para comprobar si una cadena de texto presentaba ofuscación o no se ha desarrollado un sistema de cálculo del nivel de incertidumbre basado en la entropía de Shannon, implementado en Java. Así se obtiene un conjunto de cadenas ofuscadas presentes en la aplicación Android a analizar.

• 4.2.2 ANALIZADOR DE INCERTIDUMBRE

El analizador de incertidumbre se basa en el cálculo de la entropía de Shannon. La entropía es definida como el grado de incertidumbre de una fuente de información (35).

Utilizando esto como base, se ha implementado un método de medida que permita discernir cadenas de texto ofuscadas de las que no. La entropía de Shannon es calculada usando la siguiente fórmula:

$$H(X) = \sum_{i=1}^n p(x_i) I(x_i) = \sum_{i=1}^n p(x_i) \log_b \frac{1}{p(x_i)} = - \sum_{i=1}^n p(x_i) \log_b p(x_i),$$

Ilustración 45: Fórmula Entropía de Shannon.

Esta fórmula mide la probabilidad de que una muestra seleccionada al azar de una población grande contenga exactamente n_1 individuos del tipo 1, n_2 individuos del tipo 2, ... y n_s individuos del tipo S.

En la Ilustración 46 se muestra el método implementado en Java cuya funcionalidad es calcular la entropía del parámetro que recibe.

```
public Double calculoShannonEntropia(List<Byte> values) {
    Map<Byte, Integer> map = new HashMap<Byte, Integer>();
    //Se calcula las ocurrencias de cada valor
    for (Byte sequence : values) {
        if (!map.containsKey(sequence)) {
            map.put(sequence, 0);
        }
        map.put(sequence, map.get(sequence) + 1);
    }
    //Se calcula la entropia
    Double result = 0.0;
    for (Byte sequence : map.keySet()) {
        Double frequency = (double) map.get(sequence) / values.size();
        result -= frequency * (Math.log(frequency) / Math.log(2));
    }
    return result;
}
```

Ilustración 46: Algoritmo Entropía de Shannon.

En primer lugar, se hace un recorrido a todos los bytes de la cadena de bytes introducida por parámetro, contabilizando el número de ocurrencias de cada byte.

En un segundo lugar, se calcula la frecuencia de aparición de cada uno de los bytes dentro de la cadena, que se obtiene de la división del número de apariciones de cada byte entre el número total de bytes de la cadena. Posteriormente, y empleando la fórmula de la entropía de Shannon, se multiplica la frecuencia de dicho byte por el logaritmo en base dos de la misma frecuencia. Este valor resta al resultado del byte

anterior, y así sucesivamente con todos los bytes que forman la cadena. De este modo se consigue obtener la entropía de Shannon de una cadena.

Para establecer un umbral a partir del cual se considera una cadena de texto ofuscada o no se realizó el siguiente estudio:

En primer lugar se obtuvo un diccionario de palabras, en idioma inglés, utilizando la herramienta GNU Aspell. Con esto se consiguió un listado de 138.628 palabras en claro.

En segundo lugar, se implementó un programa en lenguaje Java que generase un nuevo fichero formado por las mismas palabras pero cifradas utilizando el algoritmo DES y una clave aleatoria.

Por último, y empleando la codificación en Java de la entropía de Shannon realizada, se llevó a cabo un cálculo de la entropía de cada una de las palabras cifradas residentes en el fichero, para posteriormente calcular la media de todas las entropías obtenidas.

El resultado obtenido de calcular la media de todas las entropías, es el valor utilizado en el sistema como umbral para considerar una cadena ofuscada o no.

- **4.2.3 ANALIZADOR DE FICHEROS PRESENTES EN EL DIRECTORIO *ASSETS***

Los proyectos desarrollados para Android pueden presentar en su estructura un directorio denominado *assets*. Esta carpeta es utilizada por los desarrolladores para almacenar ficheros que son utilizados como recursos dentro de la aplicación. Su funcionalidad es similar a la del directorio *res*, pero se diferencia a este último en que los ficheros alojados en esta carpeta no se les asocian un identificador y no podrán ser modificados por la aplicación.

Este directorio es utilizado por los atacantes para almacenar código malicioso disimulando su presencia.

Este elemento del sistema es un código implementado en Java que comprueba la existencia de la carpeta *assets* en la estructura de la aplicación Android, y en tal caso almacena todas las rutas de los ficheros presentes en dicha carpeta.

- **4.2.4 ANALIZADOR DE SENTENCIAS CONDICIONALES**

Este mecanismo integrado en el analizador estático tiene como objetivo el identificar todas las sentencias condicionales presentes en el código fuente descompilado. La base de este análisis se encuentra en el flujo de ejecución y sus alternativas, con el fin de estudiar el comportamiento de todos los fragmentos de códigos posibles.

Para llevar a cabo esta tarea, se ha estudiado el formato *smali* para llevar a cabo la identificación de las siguientes sentencias condicionales:

- if-eq
- if-ne
- if-lt
- if-gt
- if-ge
- if-le
- if-eqz
- if-nez
- if-ltz
- if-gtz
- if-gez
- if-lez

Con esta información se ha realizado un código en lenguaje Java que recorre todos los ficheros con extensión *.smali*, que contienen el código fuente de la aplicación, e identifica estas sentencias para su posterior tratamiento.

• 4.2.5 ANALIZADOR DE LLAMADAS A LA API CRYPTO

Este mecanismo, integrado en el analizador estático del sistema, se centra en identificar todas las llamadas que se realizan a la API criptográfica de Java, residente en el paquete *javax.crypto*.

Se han combinado las prestaciones ofrecidas por la aplicación *Androguard* y la potencia de Java para llevar a cabo esta funcionalidad.

Se ha implementado un módulo para la herramienta *Androguard*, escrito en Python, que permite la búsqueda de las clases de la aplicación Android que hacen uso del paquete de Java *javax.crypto*. El script genera un fichero cuyo contenido son las clases que hacen uso de dicha API criptográfica.

Por otro lado, se ha implementado en Java código que, haciendo uso del fichero generado por el módulo anterior, mapea las clases que contiene. También identifica, a partir de los ficheros en formato *smali*, los métodos “doFinal()” de la clase *Cipher*. Estos métodos son los que se tratarán posteriormente para crear distintos casos de prueba.

4.3 TEST CASE MAKER

Este elemento del sistema es el motor que se encarga de generar cada uno de los casos de prueba utilizados en el análisis de *malware* ofuscado.

Esta parte del sistema está implementado en lenguaje Java y sus funcionalidades son las siguientes:

• 4.3.1 CASO DE PRUEBA: CADENA DE TEXTO OFUSCADA

Este mecanismo toma como entrada las cadenas de texto analizadas y consideradas como ofuscadas por el analizador estático. Se encarga de sustituir dicha cadena ofuscada por una cadena de texto del mismo tamaño generada aleatoriamente. Todas las apariciones de la cadena ofuscada en el código fuente son reemplazadas por la nueva cadena.

Se genera un caso de prueba por cada cadena ofuscada modificada.

- **4.3.2 CASO DE PRUEBA: FICHERO ASSETS**

Utilizando los resultados del analizador estático, esta parte del sistema toma todas las rutas de los ficheros *assets* presentes en la aplicación descompilada.

El tratamiento que hace a este tipo de archivos, es sustituir su contenido por una secuencia de bytes aleatorios del mismo tamaño que el contenido original del fichero.

Se genera un caso de prueba por cada fichero residente en la carpeta *assets* modificado.

- **4.3.3 CASO DE PRUEBA: CONDICIONALES**

Se generan cuatro tipos de casos de prueba relativos a las sentencias condicionales:

- **Condiciones invertidas:** el objetivo de este caso de prueba es invertir la condición original de todas las sentencias condicionales presentes en el código fuente. Esto se realiza identificando la condición y sustituyéndola por su condición inversa en el código.
- **Condiciones verdaderas:** la finalidad de esta implementación es que todas las instrucciones condicionales se conviertan en verdaderas. En el código Dalvik, si una sentencia condicional es cierta, salta a una instrucción señalada con una cierta etiqueta. Por lo que la implementación de este caso de prueba ha consistido en sustituir la sentencia condicional entera por una instrucción de salto a la etiqueta que indicaba la sentencia condicional.
- **Condiciones falsas:** en este caso, la finalidad de este caso de prueba es que todas las sentencias condicionales sean falsas. Para ello se ha eliminado de manera automática toda sentencia condicional con el fin de que el flujo de ejecución no sufriese ningún salto a ninguna instrucción.
- **Condiciones aleatorias:** este caso de prueba consiste en sustituir todas las sentencias condicionales por otras sentencias condicionales elegidas aleatoriamente. La implementación se ha llevado a cabo definiendo una lista formada por todas las condiciones posibles, se genera un número aleatoriamente, y la condición original es sustituida por la condición que ocupa la posición que indica dicho número en la lista. En el caso de que se haya elegido la misma condición que la original, se vuelve a generar un número aleatorio, hasta conseguir una condición distinta a la original.

- **4.3.4 CASO DE PRUEBA: LLAMADAS A LA LIBRERÍA CRYPTO**

Este elemento del sistema utiliza como entrada las clases que utilizan el paquete de Java *javax.crypto*.

El sistema se encarga de identificar en el código decompilado el método “doFinal()” de la clase *Cipher*, para modificar su resultado por una cadena de texto generada aleatoriamente.

Se modifican las instrucciones semejantes a las de la Ilustración 47:

```
invoke-virtual {v0, p0}, Ljavax/crypto/Cipher;->doFinal([B)[B
move-result-object v1
```

Ilustración 47: Instrucción llamada criptográfica.

Por instrucciones del tipo de la Ilustración 48:

```
const-string v0, "valorAleatorio"
invoke-virtual {v0}, Ljava/lang/String;->getBytes()[B\n
move-result-object v1
```

Ilustración 48: Instrucción sustitutiva a llamada criptográfica.

De esta manera, se sustituye el resultado que devuelve el método “doFinal()”, ya sea al cifrar o al descifrar, por una cadena de texto generada aleatoriamente, la cual es transformada en un array de bytes y asignada a la variable en la que se debería almacenar el resultado de la operación criptográfica.

Se realiza un caso de prueba por cada método “doFinal()” presente en el código.

4.4 COMPILADOR/REPACKAGING

Esta parte del sistema forma parte del componente Análisis de ofuscación de la arquitectura del sistema definida en el Diseño.

Su funcionalidad se centra en volver a empaquetar la aplicación Android resultante tras su paso por el Test Case Maker. La nueva aplicación, que resulta ser una modificación de la original, se vuelve a recompilar y a generar un nuevo archivo con extensión *.apk*.

La aplicación *ApkTool* permite realizar tal proceso con una de las opciones que ofrece. Pero para que la nueva aplicación pueda ser instalada en un dispositivo Android, ya sea emulado o físico, es necesario volver a firmar la aplicación con una clave pública. Ésto es debido a que la firma almacenada de la aplicación original no coincide con el de la nueva aplicación generada para el caso de prueba, ya que el código original ha sufrido modificaciones. La herramienta *Dex2Jar* ofrece la posibilidad de firmar aplicaciones

Android de una manera transparente y sencilla, sin necesidad de generar una clave pública por parte del usuario, ya que utiliza una integrada en la herramienta.

De esta manera, se obtiene una nueva aplicación, variante de la original, lista para ser instalada en un dispositivo Android para analizar su comportamiento en tiempo de ejecución.

4.5 ANALIZADOR DINÁMICO

Integrante del sistema que forma parte del componente Análisis de comportamiento definido en la arquitectura.

La implementación de este módulo se ha realizado a través de los siguientes elementos:

- **4.5.1 EMULACIÓN DE DISPOSITIVOS ANDROID**

El kit de desarrollo de software de Android contiene todas las librerías y herramientas necesarias para la simulación de un dispositivo móvil Android.

Android SDK se ha utilizado principalmente como mecanismo para proporcionar dispositivos móviles Android emulados.

Las herramientas utilizadas para generar estos dispositivos emulados presentes en este kit de desarrollo son las siguientes:

- **emulator:** proporciona dispositivos Android virtuales que se ejecutan en computadoras. Para la implementación se ha utilizado un dispositivo Android emulado cuya versión es la 2.1. La elección de esta versión fue debido a la estabilidad proporcionada con la integración con la herramienta *Droidbox*.
- **adb:** es un programa cliente-servidor que permite la comunicación con el dispositivo Android emulado. Esta herramienta se ha utilizado para la instalación de las aplicaciones Android, y para conocer en qué momento se ha inicializado por completo el sistema Android emulado, para poder trabajar con él

Haciendo uso de estas herramientas, se ha implementado un mecanismo en Java que permite generar un dispositivo Android emulado nuevo por cada caso de prueba, para evitar posibles contaminaciones de código malicioso entre distintas ejecuciones.

- **4.5.2 ANALIZADOR DE COMPORTAMIENTO**

Las herramientas utilizadas para estudiar el comportamiento de una aplicación Android han sido *Droidbox* y *Tcpdump*:

- **Droidbox:** Se han realizado modificaciones en su código fuente para adaptarlo al sistema implementado. Una de las modificaciones ha consistido en añadir una funcionalidad que permita generar un fichero de registro de la ejecución realizada, en la que se deja constancia de las operaciones realizadas y del momento en el tiempo en el que se produjeron. Otra modificación realizada ha

sido la sustitución de la herramienta *monkeyrunner*, la cual instala y ejecuta la aplicación a analizar, por la herramienta *Robotium*, que se detallará más adelante. El script que utiliza *monkeyrunner*, ha sido sustituido por uno nuevo que instala la aplicación Android, la aplicación de test y por último lanza la ejecución de esta aplicación de test automático.

- *Tcpdump*: Se ha utilizado para generar un fichero de registro de toda la actividad de red producida entre el terminal Android emulado y el exterior.

• 4.5.3 TESTER AUTOMÁTICO

El objetivo de esta parte del analizador dinámico es el de interactuar con la interfaz de usuario de la aplicación Android de manera automática.

Al utilizar *Robotium*, es necesario haber generado un proyecto de test Java que será modificado y compilado en tiempo de ejecución, ya que es necesario adaptar el código del proyecto y sus propiedades a la aplicación Android que se va a testear.

Es necesario conocer el *package* y actividades de la aplicación Android, para que al generar la aplicación Android de test interactúe con la aplicación correcta. Esta información se obtiene a través de un módulo de *Androguard* que se ha implementado para este fin. Este código genera un fichero que contiene el *package* y las actividades de la aplicación a analizar.

También, se ha implementado un módulo en Java, que lee esta información del fichero generado y la almacena. En primer lugar, este módulo en Java modifica el *package* en las propiedades del proyecto de test por el de la aplicación a analizar. En segundo lugar se genera la clase principal de test de manera automática, utilizando la información sobre las actividades obtenidas anteriormente. Esta clase incluye los siguientes algoritmos, que permiten interactuar con los siguientes tipos de elementos de la interfaz:

- **Botón Menú:**

Función testImágenes()

lanzarActividad(nombreActividad);

pulsar(menú);

FinFunción

- **Botones/ Clase Button:**

Función testBotones()

$índice \leftarrow 0;$

$lanzarActividad(nombreActividad);$

$listaBotones \leftarrow obtenerBotones ();$

Repetir

$pulsarBoton(listaBotones(índice));$

$lanzarActividad(nombreActividad);$

$índice \leftarrow índice + 1;$

Hasta Que $índice == longitud (listaBotones)$

FinFunción

- **Entradas de texto/ Clase EditText:**

Función testEditText()

$i \leftarrow 0;$

$j \leftarrow 0$

$lanzarActividad(nombreActividad);$

$listaBotones \leftarrow obtenerBotones ();$

$listaEditText \leftarrow obtenerEditTex ();$

Repetir

$i \leftarrow i + 1;$

Repetir

$rellenarEditText(texto,i);$

$pulsarBoton(listaBotones(j));$

$lanzarActividad(nombreActividad);$

$j \leftarrow j + 1;$

Hasta Que $j == longitud (listaBotones)$

Hasta Que $i == longitud (listaEditTexts);$

FinFunción

- **Imágenes/ Clase ImageView:**

Función testImágenes()

índice $\leftarrow 0$;

lanzarActividad(nombreActividad);

listaImágenes $\leftarrow obtenerImágenes()$;

Repetir

pulsarImagen(listaImágenes(índice));

lanzarActividad(nombreActividad);

índice $\leftarrow índice + 1$;

Hasta Que *índice* == longitud (*listaImágenes*)

FinFunción

Este proyecto es compilado mientras el programa principal está en funcionamiento, generando una aplicación de test lista para ser instalada y ejecutada por *Droidbox*.

- **4.5.4 ANÁLISIS INICIAL Y ALGORITMO DE OPTIMIZACIÓN DE CASOS DE PRUEBA**

- **Análisis inicial:** esta parte del sistema es la primera que se lleva a cabo. Se trata de estudiar el comportamiento de la aplicación original, sin que haya sufrido modificación alguna. Para ello se hace uso de las mismas herramientas utilizadas en los casos de prueba para el análisis dinámico de las aplicaciones Android.
- **Algoritmo de optimización del número de casos de prueba:** como el número de cadenas ofuscadas identificadas y el número de ficheros presentes en la carpeta *assets* puede resultar muy elevado, provocando un alto número de casos de prueba lo que supondría un mayor coste en tiempo, se ha decidido diseñar un algoritmo que permita optimizar el número de casos de prueba a ejecutar. Su funcionamiento se basa en la idea de que si el resultado tras un análisis de comportamiento de una cadena ofuscada o fichero *assets* es similar al del anterior caso de prueba, se pase al elemento que ocupe la posición $n/2$ de las cadenas o ficheros *assets* que falten por analizar, donde n representa el total de cadenas o ficheros restantes.

Si entre cada caso de prueba se obtiene resultados semejantes por cada tipo de operación, se considera el último resultado igual al anterior, y se salta a la cadena o fichero *assets* que se encuentre en la mitad de los restantes.

4.6 DETECCIÓN

Esta parte del sistema forma parte del componente Detección de malware ofuscado definido en la arquitectura del sistema.

La implementación se ha llevado a cabo utilizando la herramienta *tcsnark* y un módulo implementado en lenguaje Java. Por cada aplicación analizada se genera un fichero del tipo tabla de cálculo, en el que se registran el identificador y tipo del caso de prueba realizado, y el registro de todas las operaciones realizadas a modo de histograma.

La herramienta *tshark*, destinada para analizar tráfico de red, posee la capacidad de analizar ficheros con extensión *.pcap*, que se corresponden con los ficheros de registro generados por *Tcpdump*.

También se ha realizado la implementación en Java de una funcionalidad que se encarga de mapear los resultados de todos los ficheros de registro generados por las herramientas de análisis dinámicos: *Droidbox*, *Tcpdump* y *tshark*.

Utilizando toda esta información recolectada en las ejecuciones de los casos de prueba, se ha diseñado un algoritmo que analiza toda esta información contrastándola con los resultados obtenidos en los resultados iniciales, en los que se estudia el comportamiento ordinario de la aplicación.

En primer lugar se realizan dos análisis iniciales de la aplicación sin modificar, al inicio del programa. Todos los resultados relativos al comportamiento quedan registrados en el fichero de resultados generado. Por cada uno de los campos del fichero, que representa un tipo de operación, se han obtenido dos resultados, uno por cada análisis inicial. De cada campo se obtiene un ratio de variación, que se calcula del siguiente modo:

1. Calcular la media del número de operaciones por cada tipo.
2. Calcular el porcentaje de variación dividiendo el resultado de mayor número entre el menor por cada tipo de operación.
3. Calcular el ratio de estimación de error tolerable estableciendo un límite superior e inferior. El límite superior se calcula multiplicando la media por el porcentaje de variación y el límite inferior dividiendo la media por el mismo porcentaje.

De esta manera se establece un umbral en el que si el resultado del caso de prueba es superior a tantas veces la media como indica el porcentaje de variación, o es tantas veces inferior a la media como indica el mismo porcentaje, se considera que el artefacto modificado influye en el comportamiento funcional malicioso de la aplicación. Si por el contrario los resultados obtenidos se encuentran dentro de este umbral, se considera que el artefacto modificado no influye en la funcionalidad de la aplicación infectada, por lo que dicho elemento no es considerado como una pieza que el *malware* utilice para sus fines maliciosos.

CAPÍTULO 5

EVALUACIÓN

En este capítulo se detallarán los resultados obtenidos tras analizar un conjunto de aplicaciones Android utilizando el sistema desarrollado.

Se expondrá una evaluación final en la que el sistema desarrollado analizará un conjunto de aplicaciones de distintas familias de *malware*. En esta sección se presentará un conjunto de datos introductorios a nivel global relativos al comportamiento de las aplicaciones respecto a su funcionalidad original, para finalizar exponiendo los niveles de artefactos ofuscados detectados que presentan algún tipo de funcionalidad maliciosa.

Para llevar a cabo este estudio, se ha elegido un conjunto de 215 aplicaciones Android que presentan algún tipo de *malware*. Estas aplicaciones están agrupadas en distintas familias, según su comportamiento malicioso y las técnicas empleadas para explotar las vulnerabilidades presentes en el sistema Android. Para realizar esta labor se ha utilizada la colección de aplicaciones Android maliciosas utilizadas en el estudio *Dissecting Android Malware: Characterization and Evolution* (1). El estudio se ha centrado en familias que utilizan artefactos ofuscados para llevar a cabo su funcionalidad software maliciosa, siendo las estudiadas las siguientes:

- **ADDRD**: 22 aplicaciones analizadas.
- **AnserverBot**: 135 aplicaciones analizadas.
- **DroidKungFu**: 16 aplicaciones analizadas.
- **GingerMaster**: 4 aplicaciones analizadas.
- **Geinimi**: 6 aplicaciones analizadas.
- **jSMShider**: 16 aplicaciones analizadas.
- **Pjapps**: 16 aplicaciones analizadas.

5.1 ESTUDIO DEL COMPORTAMIENTO

El estudio sobre el comportamiento resultante de la evaluación realizada, utilizando el sistema desarrollado, se detalla en el [Anexo II](#) del presente documento.

El motor de detección desarrollado se basa en el análisis de la variabilidad del comportamiento ordinario de la aplicación respecto al obtenido en cada uno de los casos de prueba realizado.

En los resultados analizados sobre el comportamiento se ha apreciado que ciertos artefactos producen una mayor desviación en el comportamiento normal de la aplicación que otros, como ocurre con las cadenas de texto ofuscadas. También, se ha observado que ciertos tipos de actividad son más susceptibles a la variación dependiendo del tipo de caso de prueba, como en el caso de la actividad criptográfica respecto a los casos de prueba centrados en las llamadas a APIs criptográficas.

También se ha detectado que existe un importante porcentaje de aplicaciones analizadas que no se han obtenido resultados suficientes sobre su comportamiento

ordinario. Esto dificulta la detección de artefactos con funcionalidad ofuscada de manera automática.

5.2 RESULTADOS DE DETECCIÓN

En este apartado se van a presentar los niveles de artefactos con funcionalidad software ofuscada que ha detectado el sistema desarrollado.

En la Ilustración 49 se muestra el porcentaje de aplicaciones analizadas, por familia, en las que se han detectado algún artefacto con funcionalidad ofuscada, que ha afectado a el comportamiento ordinario de la aplicación infectada.

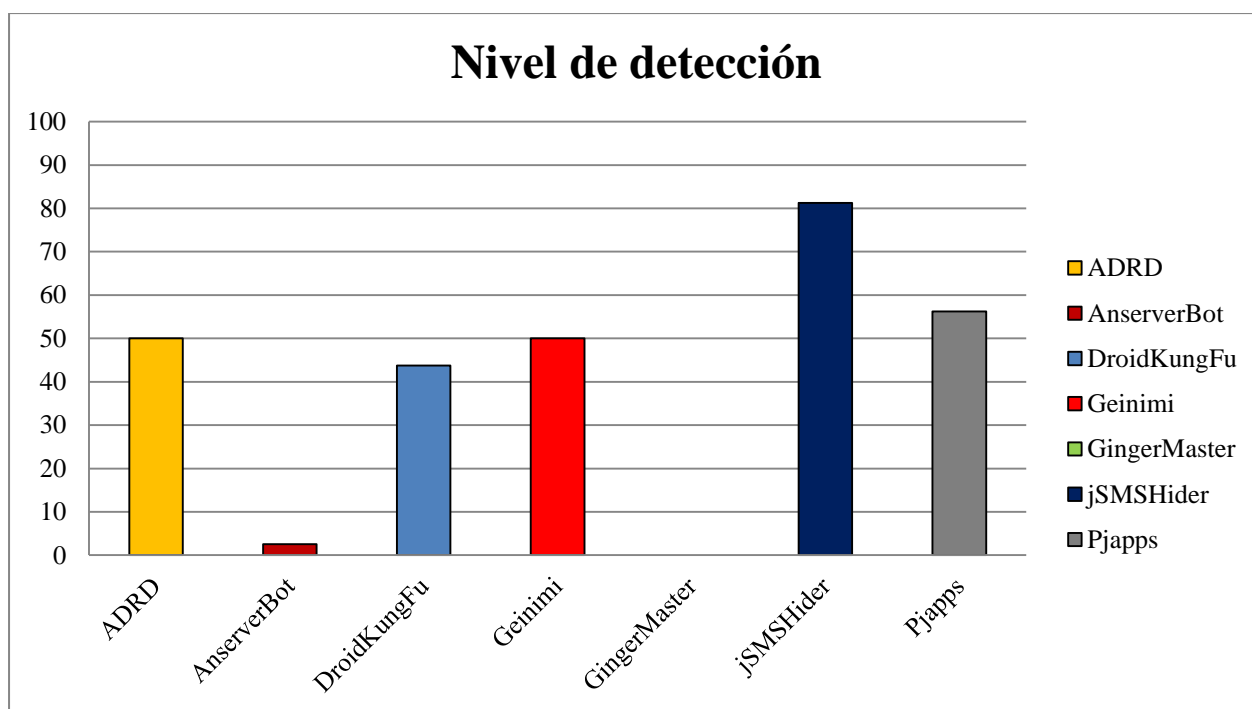


Ilustración 49: Nivel de detección.

La gráfica presenta el porcentaje de aplicaciones analizadas de cada familia de *malware* en la que se ha detectado artefactos con funcionalidad software ofuscada.

En los resultados obtenidos se muestra que del total de las aplicaciones de la familia ADRD, en un 50% se han producido detecciones de artefactos ofuscados con comportamiento malicioso. Para la familia AnserverBot este resultado ha sido de un 2,52%, mientras que para la familia DroidKungFu el resultado ha sido de un 43,75%. Para las familias Geinimi, jSMShider y Pjapps los resultados de detección obtenidos han sido de un 50%, 81,25% y 56,25% respectivamente. El sistema no ha sido capaz de detectar artefactos ofuscados maliciosos en las cuatro aplicaciones de la familia GingerMaster analizadas.

A continuación se detallan las particularidades de cada familia analizada:

- ADRD

Del conjunto de aplicaciones analizadas pertenecientes a la familia ADRD, se han detectado 70 elementos que presentan funcionalidad ofuscada. De estos artefactos, un 55,71% se han detectado al modificar cadenas de texto, un 30% al modificar el flujo de la ejecución modificando las condiciones y un 14,29% han sido encontradas en casos de prueba relacionados con APIs criptográficas:

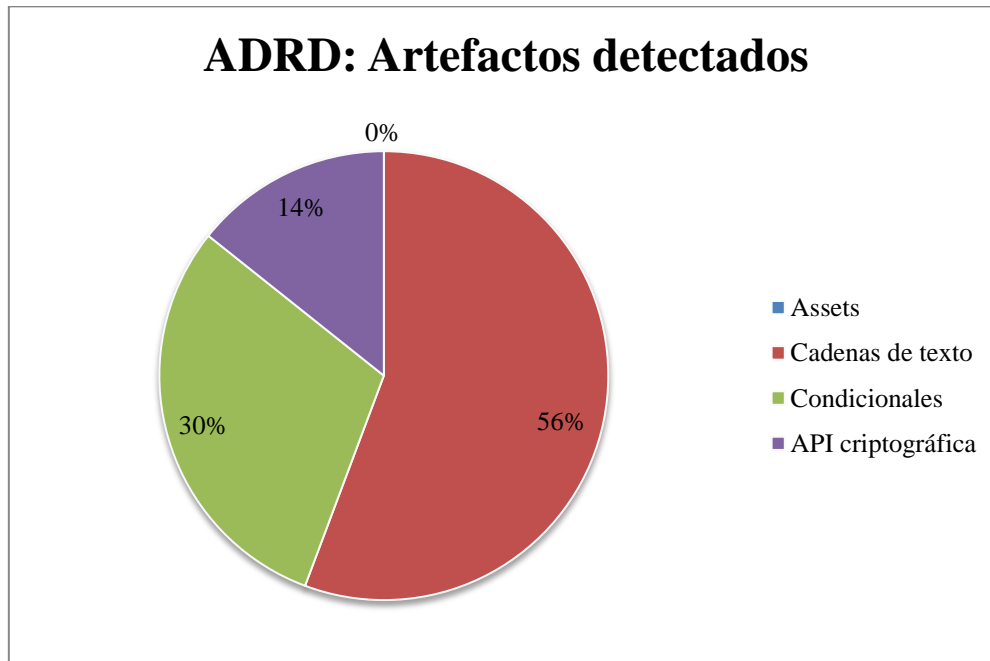


Ilustración 50: ADRD: Artefactos detectados.

- ANSERVERBOT

Se han identificado 8 artefactos con funcionalidad ofuscada tras analizar el grupo de aplicaciones de esta familia. De estos artefactos, un 15,38% se corresponde con ofuscación en ficheros *assets*, un 69,24% en cadenas de texto ofuscadas, un 7,69% han sido detectadas al modificar el flujo de ejecución de la aplicación y otro 7,69% han sido detectados en llamadas a librerías criptográficas.

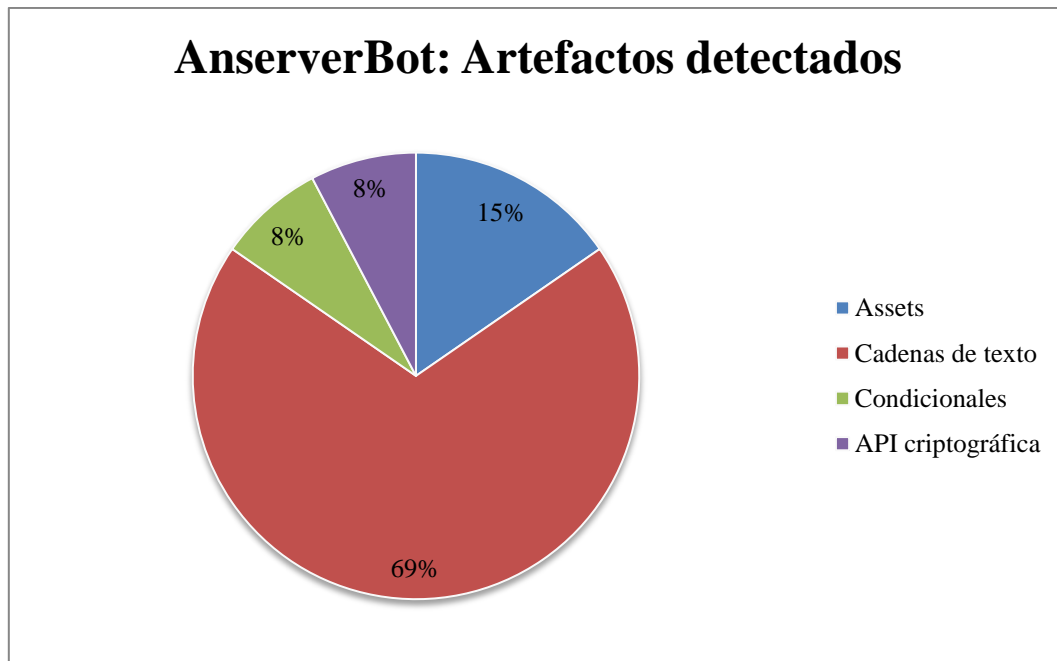


Ilustración 51: AnserverBot: Artefactos detectados.

- **DROIDKUNGFU**

Se han detectado 45 elementos con funcionalidad ofuscada en las aplicaciones de la familia DroidKungFu analizadas con el sistema desarrollado. De los elementos ofuscados un 31,81% se corresponde con ficheros almacenados en el directorio *assets*, un 40,9% a cadenas de texto, un 20,45% han sido detectadas en mutaciones de sentencias condicionales y un 6,84% en llamadas a APIs criptográficas.

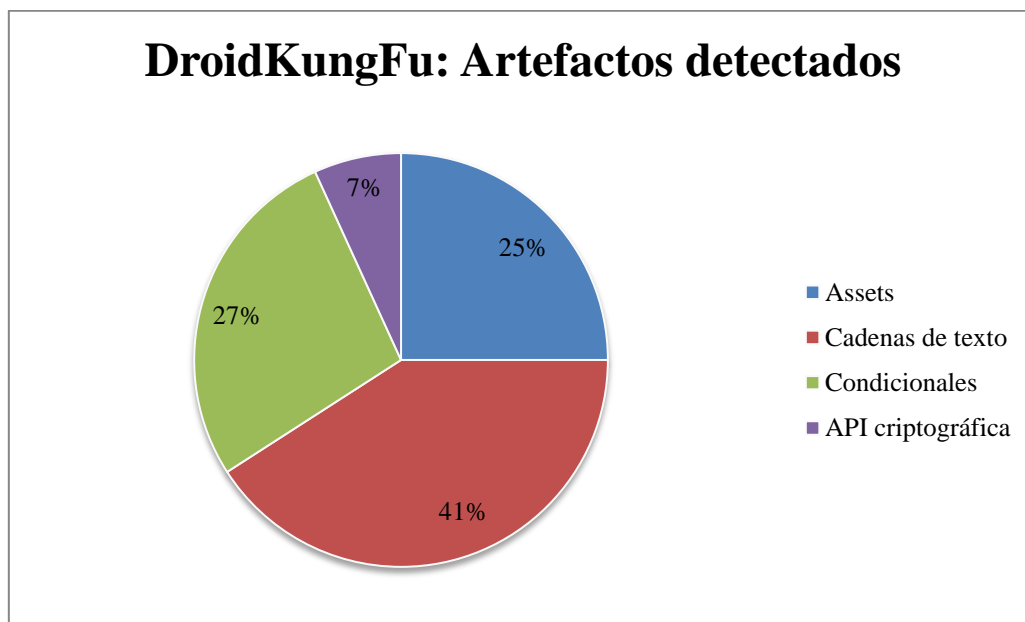


Ilustración 52: DroidKungFu: Artefactos detectados.

- **GEINIMI**

Se han identificado 19 artefactos con funcionalidad ofuscada en esta familia de *malware*. Un 21,05% ha sido detectado en casos de prueba del tipo Cadenas ofuscadas, un 52,63% en casos de prueba relacionados con cadenas de texto ofuscadas y un 26,32% en llamadas a APIs criptográficas.

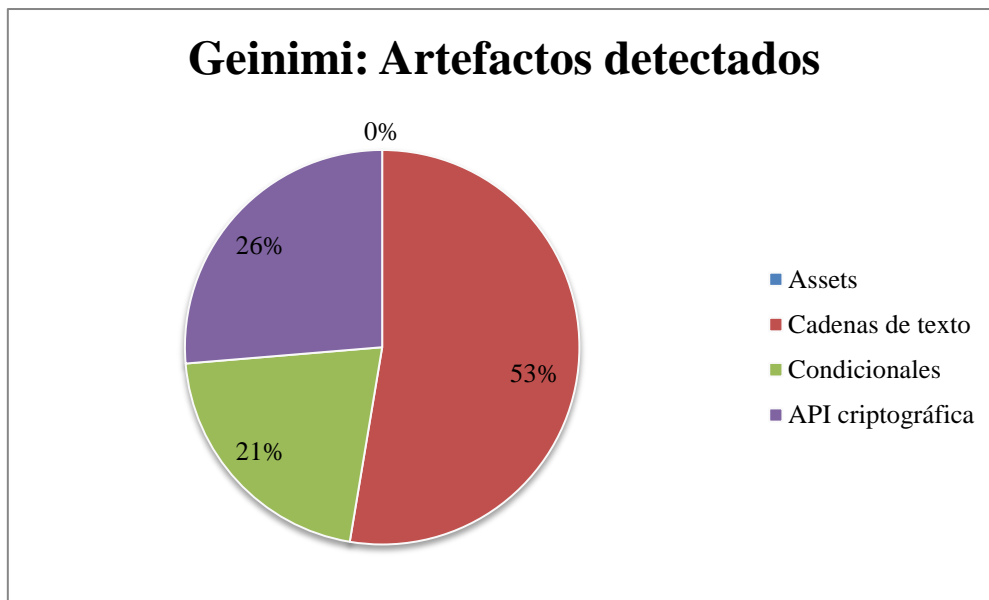


Ilustración 53: Geinimi: Artefactos detectados.

- **GINGERMASTER**

De las cuatro aplicaciones analizadas en ninguna se ha producido detección de artefactos con funcionalidad escondida por parte del sistema.

- **JSMSHIDER**

Los artefactos con funcionalidad ofuscada identificados en el conjunto de aplicaciones de la familia jSMShider han sido 25. El 50% de los elementos han sido en casos de prueba de mutación de cadenas de texto, un 4,17% en casos de prueba del tipo condicional y un 45,83% en casos de prueba del tipo llamadas a APIs criptográficas. No se han producido detecciones de elementos ofuscados en *assets* porque este tipo de familia no utiliza este tipo de ofuscación para llevar a cabo sus fines maliciosos.

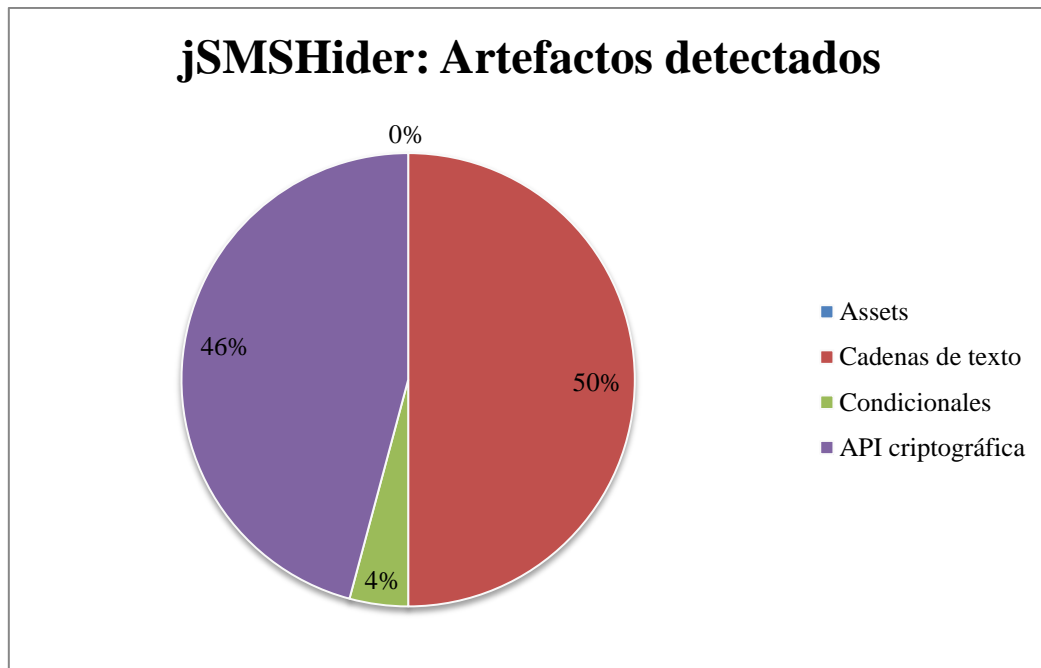


Ilustración 54: jSMShider: Artefactos detectados.

- **PJAPPS**

El total de elementos con funcionalidad ofuscado han sido 88 del total de aplicaciones analizadas de la familia Pjapps. De estos artefactos, el 28,91% se ha detectado en ficheros *assets*, un 45,35% a través de las cadenas ofuscadas mutadas y el 26,74% restante a través de modificaciones del flujo de ejecución de la aplicación. No se han detectado artefactos con funcionalidad ofuscada a través de llamadas a librerías del tipo criptográfico.

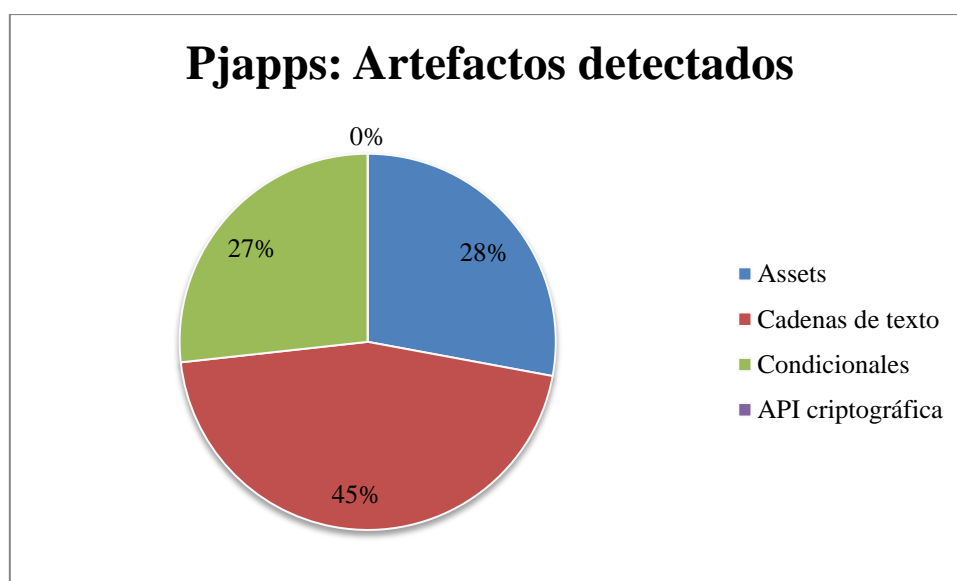


Ilustración 55: Pjapps: Artefactos detectados.

5.3 DISCUSIÓN

Los resultados obtenidos concuerdan, en su mayor parte, con el [Estudio previo](#), situado en el Capítulo 2: Análisis del presente documento.

En el caso de la familia DroidKungFu respecto al Estudio previo, se analizó manualmente que presentaban artefactos con funcionalidad ofuscada tanto en ficheros *assets* (exploit *ratc*) como variables ofuscadas en el código fuente. En los resultados obtenidos se observan que los porcentajes de detección más altos se han conseguido en casos de prueba centrados en analizar este tipo de elementos.

La familia Pjapps se caracteriza por presentar en su código fuente direcciones cifradas a servidores C&C. Los resultados obtenidos muestran que los casos de prueba centrados en analizar cadenas de texto ofuscadas ha sido la que mayor porcentaje de detección ha presentado. Estos resultados concuerdan con los resultados obtenidos en el caso de prueba realizado, de manera manual, en el análisis centrado en esta familia.

Por el contrario, en las familias GingerMaster y AnserverBot no se han obtenido unos resultados tan similares. Principalmente, esto ha sido debido a que el comportamiento ordinario detectado en las aplicaciones de estas familias no ha sido lo suficientemente bueno, ya que en su mayoría solo se han producido operaciones de lectura y escritura en disco solamente.

Tanto GingerMaster como AnserverBot hacen uso de ficheros ofuscados que se ejecutan cuando la aplicación está en funcionamiento. AnserverBot (5) instala dos aplicaciones, que ejecutan la funcionalidad maliciosa, escondidas tras ficheros *assets*. La no ejecución de estas aplicaciones durante el análisis puede haber producido los resultados obtenidos por esta familia. Mientras, GingerMaster (1) utiliza scripts y *exploits* que aparentan ser imágenes, para poder escalar privilegios dentro del sistema. Los resultados obtenidos pueden deberse a que el tiempo de ejecución de la aplicación para estudiar su comportamiento haya sido insuficiente, no permitiendo al *malware* escalar los privilegios necesarios para seguir ejecutando sus acciones maliciosas.

El resto de familias analizadas utilizan técnicas de ofuscación similares al de las familias anteriores (3).

La familia ADRD, también conocida como HongTouTou, utiliza la ofuscación para cifrar las direcciones a servidores de C&C. Los resultados obtenidos muestran que el mayor porcentaje de detecciones se han producido en casos de prueba en el que se manipulan cadenas de texto ofuscadas.

La familia jSMShider se comunica con servidores C&C utilizando técnicas de cifrado, por eso los altos niveles de operaciones criptográficas presentes en su comportamiento ordinario. Los resultados obtenidos presentan que el mayor porcentaje de niveles detectados se han realizado en casos de prueba relativos a la modificación de cadenas de texto ofuscadas y de llamadas a APIs criptográficas.

La familia Geinimi también utiliza la ofuscación a través de técnicas de cifrado para intercambiar información con servidores C&C. Y nuevamente los resultados obtenidos muestran que los mayores porcentajes de detección se han realizado en casos de prueba destinados a analizar cadenas de texto ofuscadas y llamadas a APIs criptográficas.

En resumen, se aprecia que los mayores niveles de detección de artefactos con funcionalidad ofuscada se han encontrado en variables que presentan ofuscación. En el estudio *Dissecting Android Malware: Characterization and Evolution* (1) se presenta como principal técnica de ofuscación utilizada por la mayoría de las familias de *malware* el uso de variables ofuscadas, como ocurre en las familias Pjapps, DroidKungFu y Geinimi, entre otras, que cifran las direcciones de servidores de C&C. Esta conclusión concuerda con los resultados obtenidos en la evaluación realizada, en el que los niveles de detección más altos se han obtenido en casos de prueba centrados en analizar cadenas de texto ofuscadas en el código fuente de la aplicación.

Otro tipo de ofuscación identificado en el estudio anterior (1) se trata en la ofuscación de código malicioso a través de ficheros presentes en el directorio de recursos *assets*. Este tipo de ofuscación es utilizado por un menor número de familias, entre las cuales se encuentran DroidKungFu (presenta root exploits como el fichero *ratc*), la familia GingerMaster (ofusca sus root exploits para que fingen ser imágenes) y AnserverBot (presenta dos ficheros que aparentan ser bases de datos, pero en realidad son aplicaciones con funcionalidad maliciosa que son instaladas mientras la aplicación está en ejecución). Los resultados obtenidos en la evaluación coinciden con este hecho, ya que se han obtenido resultados de detección analizando ficheros *assets* en dos de las tres familias analizadas que presentaban este tipo de ofuscación.

Las familias de *malware* que conectan con servidores C&C, formando botnets, como son Pjapps, DroidKungFu o Geinimi, entre otras (1), utilizan técnicas de cifrado para transmitir información a través de la red. La información que se transmite del dispositivo infectado al servidor, en su mayoría datos sensibles del usuario, es cifrada en el dispositivo y transmitida a la red. Cuando esta información es recibida en el servidor de comando y control, está es descifrada y utilizada por el servidor. También el servidor envía información al dispositivo, órdenes en la mayoría de los casos, que es cifrada en su origen y descifrada por el dispositivo móvil. Con los casos de prueba centrados en analizar las APIs criptográficas, se altera esta funcionalidad maliciosa, ya que la información transmitida entre el dispositivo móvil infectado y el servidor C&C es sustituida por bytes aleatorios. Con esta idea resultados de detección al manipular llamadas a APIs criptográficas en familias que presentan este comportamiento malicioso.

En cuanto a los casos de prueba centrados en condicionales, los resultados indican que las aplicaciones analizadas presentan un bajo porcentaje de detección, debido a que el *malware* actual no oculta funcionalidad en segmentos de código que se activen condicionalmente. Por este motivo, este tipo de caso de prueba no presenta una relevancia como puede presentar la ofuscación de variables o de ficheros *assets*, pero en

un futuro puede convertirse en una nueva técnica de ofuscación de funcionalidad maliciosa.

En la mayoría de las familias analizadas, se han obtenido niveles de detección en torno al 50%, destacando el alto porcentaje de detección en aplicaciones de la familia jSMShider. También ha habido dos familias de *malware* (AnserverBot y GingerMaster) en los que los niveles de detección han sido notoriamente inferiores al resto de familias. Este hecho puede ser debido a la complejidad del *malware*, el cual necesite de alguna condición inicial que active la funcionalidad maliciosa, o que la duración empleado en los análisis dinámicos haya sido insuficiente, no dando tiempo al código malicioso a actuar.

Con los resultados obtenidos, se puede observar que ciertos tipos de modificación afectan más a unos tipos de *malware* que a otros. Aquellos tipos de *malware* que utilicen exploits y scripts para escalar privilegios en el sistema, se ven más afectados por casos de prueba centrados en analizar ficheros *assets*, debido a que la mayoría de estos ficheros son camuflados como recursos en este directorio. Por otro lado, aquellas aplicaciones maliciosas que conectan el dispositivo infectado a una botnet, ven su funcionalidad maliciosa comprometida mayormente por casos de prueba en que se alteran los resultados de operaciones criptográficas y en que se modifican variables ofuscadas. Por lo que, dependiendo del tipo de modificación realizado y del comportamiento resultante, se puede llegar a conocer al tipo de *malware* presente en la aplicación analizada.

Como conclusión final, destacar que futuros estudios que sigan esta dirección, podrían incorporar nuevos de tipos de modificaciones, añadiendo nuevos tipos de casos de prueba, que permitan distinguir entre familias según el tipo de *malware* analizado.

CAPÍTULO 6

CONCLUSIONES

En este capítulo se presentan las conclusiones obtenidas tras la finalización del Trabajo de Fin de Grado, así como una serie de posibles trabajos futuros en base a este proyecto.

6.1 CONCLUSIONES GENERALES

El objetivo principal de este TFG es el de desarrollar un sistema para aplicaciones Android que, de manera automática, sea capaz de detectar artefactos ofuscados que escondan tras de sí una funcionalidad software maliciosa.

Este procedimiento supone un gran trabajo y esfuerzo para un analista de seguridad, ya que las aplicaciones presentan en su estructura un gran número de ficheros, directorios, variables y recursos. El inspeccionar de forma manual todos estos elementos en búsqueda de artefactos ofuscados, además de enfrentarse al reto de aclarar dicha información para determinar si su contenido original contiene algún tipo de funcionalidad que pueda cambiar el comportamiento del *malware*, supone un trabajo de una complejidad enorme para una persona.

A través del método propuesto en este proyecto, se ha podido analizar, diseñar e implementar un sistema que automatice esta tarea. Y a la vista de los resultados obtenidos en el capítulo de Evaluación, se puede observar cómo se puede llevar a cabo la identificación de artefactos ofuscados y la interpretación de la funcionalidad que esconden de una manera automática. De esta manera, se consigue reducir el tiempo y la complejidad empleado por un analista de seguridad en desempeñar este tipo de tareas, ahorrando así costes de tiempo y económicos, y poder emplearlo en otro tipo de tareas que lo requieran.

Finalmente, se puede concluir que se han alcanzado todos los objetivos propuestos en el presente Trabajo de Fin de Grado, consiguiendo desarrollar satisfactoriamente un sistema capaz de realizar esta labor de una manera automática.

6.2 LÍNEAS FUTURAS

En esta sección se numeran las posibles ampliaciones del sistema desarrollado que se han identificado.

6.2.1 INTEGRACIÓN CON MERCADOS DE APLICACIONES ANDROID

Puesto que el sistema desarrollado se centra en la detección de código malicioso en aplicaciones Android, se podría desarrollar una variante que fuese integrada con mercados de aplicaciones Android, tanto como los mercados legítimos como alternativos. En los mercados oficiales serviría de apoyo para los analistas de seguridad encargados de detectar *malware* en las aplicaciones publicadas por los desarrolladores.

Mientras, en los mercados alternativos, los cuales presentan medidas de seguridad pobres, presentan uno de los principales focos de expansión de *malware* en dispositivos Android. Por lo que con esta integración se podría conseguir añadir una barrera de seguridad en este tipo de mercados, y paliar, en la medida de lo posible, la expansión de código malicioso en este tipo de dispositivos móviles.

6.2.2 INTEGRACIÓN CON MOTOR ANTIVIRUS

La base empleada para el desarrollo del sistema presentado en este TFG, puede ser utilizada para extender la funcionalidad de motores antivirus centrados en el sistema Android. Este componente del motor del antivirus se centraría en *malware* que se base en utilizar las técnicas de ofuscación analizadas y estudiadas en este proyecto.

6.2.3 ADAPTACIÓN A NUEVAS TÉCNICAS OFUSCADAS MALICIOSAS

El desarrollo de *malware* por parte de los atacantes está en constante crecimiento. Con el tiempo las técnicas empleadas por los atacantes varían o desarrollan nuevos métodos. Se podría actualizar el sistema desarrollado añadiendo nuevos casos de prueba basados en las nuevas variantes de técnicas de ofuscación maliciosas que puedan aparecer en el futuro.

6.2.4 INCORPORACIÓN DE TÉCNICAS BASADAS EN DATA MINING

La minería de datos (Data Mining) se trata de un campo de las ciencias de la computación referido al proceso que pretende descubrir patrones en grandes volúmenes de conjuntos de datos. El proceso de la minería de datos podría ser útil para analizar los datos resultantes obtenidos con el sistema desarrollado tras estudiar grandes conjuntos de aplicaciones. Permitiría realizar agrupaciones de aplicaciones que presentan resultados similares, permitiendo identificar a la familia de *malware* a la que pertenecen, o detecciones de nuevos tipos de *malware* ofuscado que presenten comportamientos novedosos.

CAPÍTULO 7

GESTIÓN DEL PROYECTO

En este capítulo se precisan todos los aspectos referentes a la gestión del proyecto, detallando la planificación del trabajo, los medios técnicos empleados para el desarrollo del TFG y el análisis económico del mismo.

7.1 PLANIFICACIÓN DEL TRABAJO

En esta sección se especifica la planificación inicial elaborada al inicio del proyecto, la cual presenta estimaciones temporales, y el seguimiento real del proyecto. Tanto la planificación inicial como el desarrollo real del proyecto se presentan en sendos diagramas de Gantt. También se ha incluido un estudio de las desviaciones temporales producidas entre la planificación inicial y el desarrollo real, indicando las razones por las que existe divergencia entre ellos.

El ciclo de vida utilizado para el desarrollo de este proyecto ha sido el ciclo de vida en cascada, que consta de las siguientes fases:

- **Especulación:** período en el que se identifican los objetivos del proyecto, se define una planificación inicial y en la que se prepara y estudia el sistema utilizado durante el desarrollo del TFG.
- **Análisis:** etapa en la que se lleva a cabo la extracción y refinamiento de los requisitos del sistema. También se lleva a cabo la especificación de los mismos elaborando los casos de uso pertinentes. Igualmente se realizan estudios iniciales relacionados con el objetivo del proyecto.
- **Diseño:** fase en la que se define la arquitectura del sistema a desarrollar, definiendo sus distintos componentes y los nexos que los unen, así como la especificación de las clases que lo forman.
- **Implementación:** período en el que se reduce el diseño elaborado a código.
- **Evaluación:** etapa en la que se realiza un estudio de detección de *malware* ofuscado empleando el sistema desarrollado con un conjunto de aplicaciones.
- **Documentación:** fase que engloba la redacción, edición y revisión de la memoria del proyecto.

7.1.1 PLANIFICACIÓN INICIAL

En esta sección se presenta la planificación inicial, realizada mediante estimaciones temporales.

Para la elaboración de esta planificación se han establecido jornadas laborales de 4 horas diarias, incluyendo el sábado como un día laboral. El motivo de la elección de esta planificación se debe a la necesidad de compaginar el desarrollo del proyecto con otros estudios y compromisos.

La planificación inicial del proyecto abarca desde el 11 de febrero de 2013 hasta el 11 de julio de 2013.

A continuación se exponen el diagrama de Gantt con la planificación inicial realizada (Ilustración 56) y el calendario previsto para cada tarea de cada una de las fases (Tabla 113):

<i>Nombre de tarea</i>	<i>Duración</i>	<i>Comienzo</i>	<i>Fin</i>
Trabajo Fin de Grado	130 días	lun 11/02/13	jue 11/07/13
Especulación	9 días	lun 11/02/13	mié 20/02/13
Definir Objetivos	2 días	lun 11/02/13	mar 12/02/13
Realizar Planificación	2 días	mar 12/02/13	mié 13/02/13
Preparación del sistema	5 días	vie 15/02/13	mié 20/02/13
Instalación Santoku	1 día	vie 15/02/13	vie 15/02/13
Instalación DroidBox	2 horas	sáb 16/02/13	sáb 16/02/13
Aprendizaje DroidBox	2 horas	sáb 16/02/13	sáb 16/02/13
Aprendizaje APKTool	1 día	lun 18/02/13	lun 18/02/13
Aprendizaje Dex2Jar	1 día	mar 19/02/13	mar 19/02/13
Aprendizaje Androguard	1 día	mié 20/02/13	mié 20/02/13
Análisis	8 días	jue 21/02/13	vie 01/03/13
Extracción de Requisitos	1 día	jue 21/02/13	jue 21/02/13
Definición de requisitos	2 días	vie 22/02/13	sáb 23/02/13
Definición de casos de uso	1 día	lun 25/02/13	lun 25/02/13
Estudio malware ofuscado	4 días	mar 26/02/13	vie 01/03/13
Diseño	6 días	mar 05/03/13	lun 11/03/13
Diseño de arquitectura	3 días	mar 05/03/13	jue 07/03/13
Diseño de clases	3 días	vie 08/03/13	lun 11/03/13
Implementación	50 días	mar 02/04/13	mié 29/05/13
Implementación en Python	5 días	mar 02/04/13	sáb 06/04/13
Implementación en Java	30 días	lun 08/04/13	sáb 11/05/13
Pruebas	15 días	lun 13/05/13	mié 29/05/13
Evaluación	13 días	mié 29/05/13	mié 12/06/13
Instalación en entorno cloud	3 días	mié 29/05/13	vie 31/05/13
Ejecución de la evaluación	10 días	sáb 01/06/13	mié 12/06/13
Documentación	20 días	mié 12/06/13	jue 04/07/13
Redacción	16 días	mié 12/06/13	sáb 29/06/13
Edición	1 día	lun 01/07/13	lun 01/07/13
Corrección	3 días	mar 02/07/13	jue 04/07/13

Tabla 113: Planificación inicial.

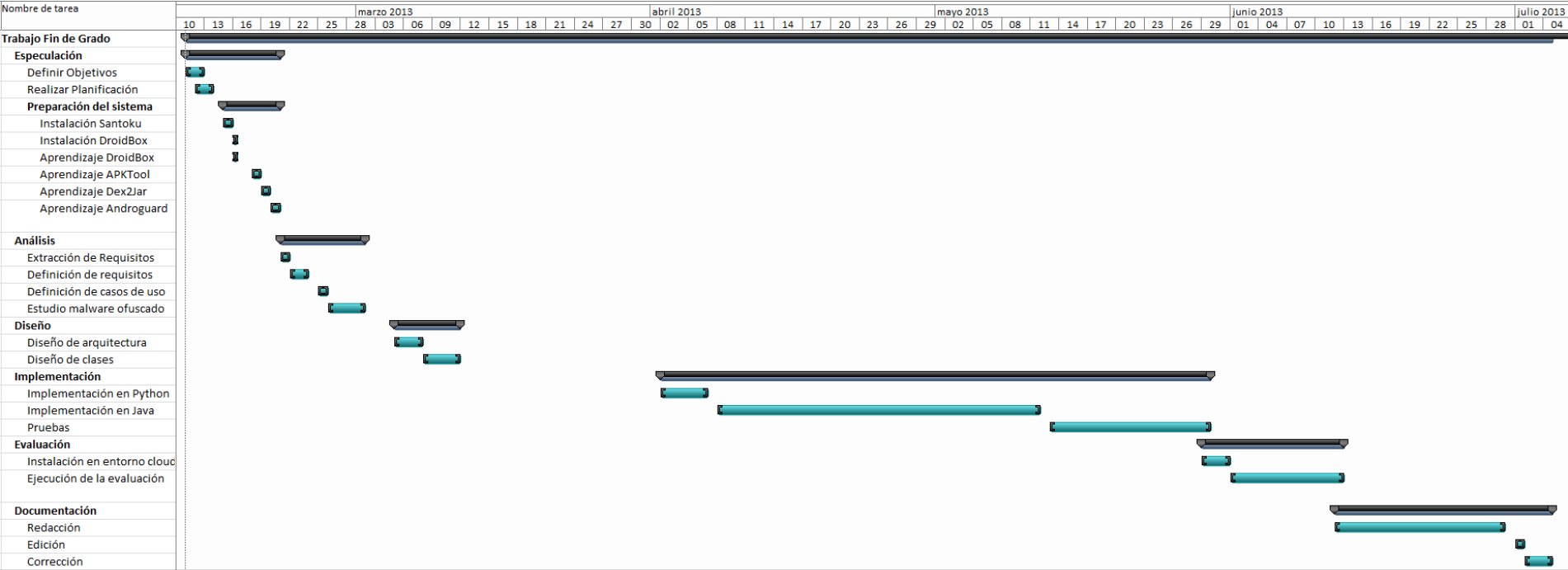


Ilustración 56: Diagrama de Gantt: Planificación inicial.

7.1.2 DESARROLLO REAL DEL PROYECTO

En esta sección se presenta el desarrollo real del proyecto. También se ha realizado una comparación de éste con la planificación inicial detallada anteriormente, para estudiar las desviaciones producidas a lo largo del desarrollo del proyecto.

En la Tabla 114 se detalla la duración real de cada una de las tareas realizadas durante el desarrollo del proyecto y en la Ilustración 58 se muestra el diagrama de Gantt correspondiente. Cabe destacar que la duración total del proyecto ha sido superior en cuarenta y un días respecto a la planificación inicial realizada.

<i>Nombre de tarea</i>	<i>Duración</i>	<i>Comienzo</i>	<i>Fin</i>
Trabajo Fin de Grado	171 días	lun 11/02/13	mié 28/08/13
Especulación	9 días	lun 11/02/13	mié 20/02/13
Definir Objetivos	2 días	lun 11/02/13	mar 12/02/13
Realizar Planificación	2 días	mié 13/02/13	jue 14/02/13
Preparación del sistema	5 días	vie 15/02/13	mié 20/02/13
Instalación Santoku	1 día	vie 15/02/13	vie 15/02/13
Instalación DroidBox	2 horas	sáb 16/02/13	sáb 16/02/13
Aprendizaje DroidBox	2 horas	sáb 16/02/13	sáb 16/02/13
Aprendizaje APKTool	1 día	lun 18/02/13	lun 18/02/13
Aprendizaje Dex2Jar	1 día	mar 19/02/13	mar 19/02/13
Aprendizaje Androguard	1 día	mié 20/02/13	mié 20/02/13
Análisis	5 días	lun 04/03/13	vie 08/03/13
Extracción de Requisitos	1 día	lun 04/03/13	lun 04/03/13
Definición de requisitos	1 día	mar 05/03/13	mar 05/03/13
Definición de casos de uso	1 día	mié 06/03/13	mié 06/03/13
Estudio malware ofuscado	2 días	jue 07/03/13	vie 08/03/13
Diseño	16 días	lun 11/03/13	jue 28/03/13
Diseño de arquitectura	5 días	lun 11/03/13	vie 15/03/13
Diseño de clases	4 días	lun 25/03/13	jue 28/03/13
Implementación	68 días	lun 08/04/13	mar 25/06/13
Implementación en Python	4 días	lun 08/04/13	jue 11/04/13
Implementación en Java	42 días	lun 08/04/13	sáb 25/05/13
Pruebas	23 días	lun 27/05/13	mar 25/06/13
Evaluación	28 días	mié 24/07/13	sáb 24/08/13
Instalación en entorno cloud	13 días	mié 24/07/13	mié 07/08/13
Ejecución de la evaluación	15 días	jue 08/08/13	sáb 24/08/13
Documentación	27 días	lun 29/07/13	mié 28/08/13
Redacción de memoria	22 días	lun 29/07/13	jue 22/08/13
Edición	1 día	vie 23/08/13	vie 23/08/13
Corrección	4 días	sáb 24/08/13	mié 28/08/13

Tabla 114: Desarrollo real del proyecto.

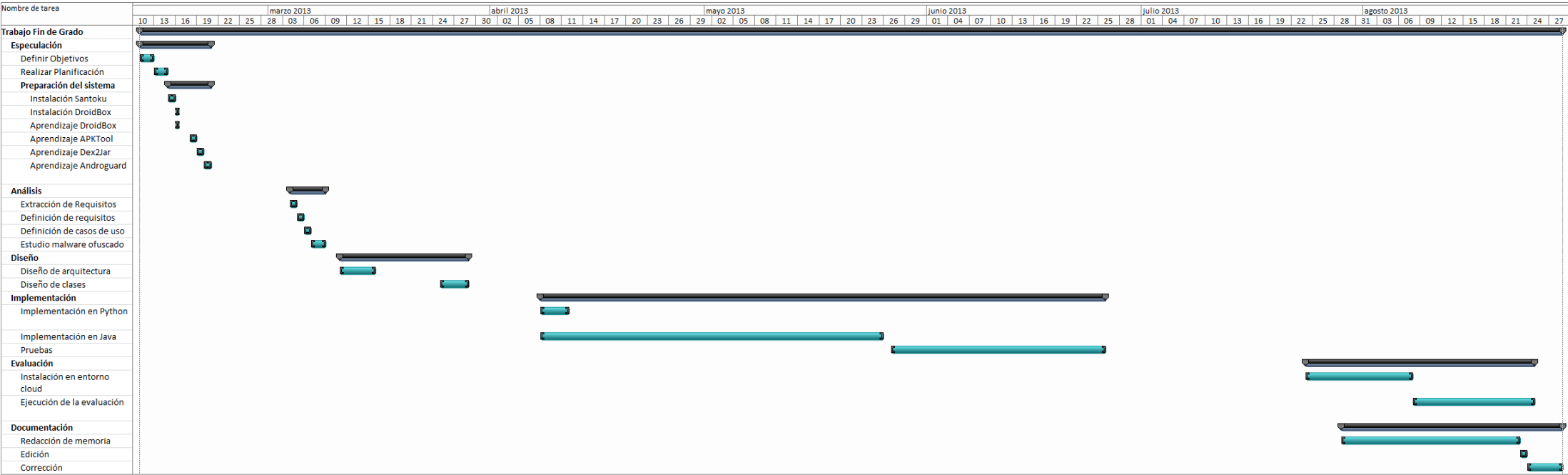


Ilustración 57: Diagrama de Gantt: Desarrollo real del proyecto.

En la Tabla 115 se muestra un análisis comparativo entre la planificación inicial estimada y el desarrollo real del proyecto.

En la tabla se puede observar que se ha producido una desviación total de un 31,53%. Las principales causas de esta desviación se atribuyen a la falta de experiencia en la planificación de proyectos, y a la aparición de problemas en las etapas de Implementación y Evaluación, que alargaron más de lo esperado la duración de dichas etapas.

<i>Nombre de tarea</i>	<i>Planificado</i>	<i>Real</i>	<i>Diferencia</i>	<i>Variación</i>
Trabajo Fin de Grado	130 días	171 días	41 días	31,53%
Especulación	9 días	9 días	0 días	0,00%
Definir Objetivos	2 días	2 días	0 días	0,00%
Realizar Planificación	2 días	2 días	0 días	0,00%
Preparación del sistema	5 días	5 días	0 días	0,00%
Instalación Santoku	1 día	1 día	0 días	0,00%
Instalación DroidBox	2 horas	2 horas	0 horas	0,00%
Aprendizaje DroidBox	2 horas	2 horas	0 horas	0,00%
Aprendizaje APKTool	1 día	1 día	0 días	0,00%
Aprendizaje Dex2Jar	1 día	1 día	0 días	0,00%
Aprendizaje Androguard	1 día	1 día	0 días	0,00%
Análisis	8 días	5 días	-3 días	-37,5%
Extracción de requisitos	1 día	1 día	0 días	0,00%
Definición de requisitos	2 días	1 día	-1 días	-50,00%
Definición de casos de uso	1 día	1 día	0 días	0,00 %
Estudio malware ofuscado	4 días	2 días	-2 días	-50,00%
Diseño	6 días	16 días	10 días	166,66 %
Diseño de arquitectura	3 días	5 días	2 días	66,66%
Diseño de clases	3 días	4 días	1 días	33,33%
Implementación	50 días	68 días	18 días	36,00%
Implementación en Python	5 días	4 días	-1 días	-20,00%
Implementación en Java	30 días	42 días	12 días	40,00%
Pruebas	15 días	23 días	8 días	53,33%
Evaluación	13 días	28 días	15 días	115,38%
Instalación en entorno cloud	3 días	13 días	10 días	333,33%
Ejecución de la evaluación	10 días	15 días	5 días	50,00%
Documentación	20 días	27 días	7 días	35,00%
Redacción	16 días	22 días	6 días	37,50%
Edición	1 día	1 día	0 días	0,00%
Corrección	3 días	4 días	1 días	33,33%

Tabla 115: Análisis de desviaciones en la planificación.

7.2 MEDIOS TÉCNICOS EMPLEADOS PARA EL PROYECTO

En la realización del TFG se ha hecho uso de herramientas software y hardware específicos.

7.2.1 HERRAMIENTAS HARDWARE

Equipo portátil:

- Acer Aspire 5740
- Micropocesorador Intel® Core™ i3 330M
- 4 GB de RAM
- 2,13 GHz
- 320 GB HDD

7.2.2 HERRAMIENTAS SOFTWARE

Sistemas operativos:

- Windows 7 Home Premium SP1 64bit
- Santoku Linux OS
- Ubuntu Desktop 12.04 LTS

Herramientas:

- Java Development Kit 7.0
- Java Runtime Environment (JRE)
- Eclipse Juno
- Python 2.7
- Android SDK
- Androguard
- APKTool
- DroidBox23
- Gedit
- Notepad ++
- VMware Player
- Tcpdump
- Wireshark
- Dex2Jar
- Tshark
- GNU Aspell
- Leafpad

Suites ofimáticas:

- Microsoft® Office™ Project 2010 Professional
- Microsoft® Office™ 2010 Professional
- LibreOffice® 3

Herramientas Web

- Dropbox
- Mozilla Firefox® 20.0.1

7.3 ANÁLISIS ECONÓMICO DEL PROYECTO

En esta sección se detalla el análisis económico del TFG realizado, incluyendo la metodología utilizada para estimar los costes, el cálculo del presupuesto inicial, el cálculo del presupuesto para el cliente y el cálculo del coste real del proyecto.

7.3.1 METODOLOGÍA DE ESTIMACIÓN DE COSTES

La estimación de costes se ha realizado teniendo en cuenta costes directos y costes indirectos, utilizando como base la plantilla para el cálculo de presupuestos proporcionada por la Universidad Carlos III de Madrid.

Los costes directos son aquellos relacionados directamente con el desarrollo del proyecto, como son los gastos de mano de obra, los equipos utilizados y las herramientas de software empleadas.

Los costes indirectos son aquellos en los que se incurre para el desarrollo del proyecto pero que no están ligados directamente con el desarrollo del sistema. Según la plantilla utilizada el valor de estos gastos se corresponderá con el 20% del valor de los costes directos.

7.3.2 PRESUPUESTO INICIAL

En esta sección se detalla el presupuesto inicial, presentando los costes presupuestados junto con el presupuesto total estimado.



UNIVERSIDAD CARLOS III DE MADRID

Escuela Politécnica Superior

ESTIMACIÓN DEL PRESUPUESTO DE PROYECTO

1. Autor:
Mario Herreros Díaz
2. Departamento:
Informática
3. Descripción del proyecto:
 - Título: Análisis de ofuscación en Apps mediante casos de prueba
 - Duración (meses): 4,3 meses
 - Tasa de costes indirectos: 20%
4. Presupuesto estimado total del proyecto (€): 22.015,50
5. Desglose presupuestario (costes directos)

PERSONAL

En este apartado se detallan los gastos de personal. Para el cálculo de estos gastos se ha tenido en cuenta la participación de una única persona. La dedicación de esta persona es de cuatro horas diarias en días laborables y sábados durante los 130 días estimados de duración del proyecto.

El coste hombre/mes se ha establecido en 2.694,39 €, que es el valor indicado en la plantilla utilizada como base para un ingeniero. También se ha tenido en cuenta el importe a pagar a la Seguridad Social, el cual es de un 28,3% del sueldo bruto del trabajador.

Apellidos y nombre	Categoría	Dedicación ^{a)} Persona/ mes	Coste Persona/ mes	Coste bruto	Seguridad Social	Coste total
Herreros Díaz, Mario	Ingeniero	4,3	2.694,39 €	2.694,39 €	762,51€	3.456,90 €
Total:						14.864,67 €

- a) 1 Hombre mes = 131.25 horas. Máximo anual de dedicación de 12 hombres mes (1.575 horas)
Máximo anual para PDI de la Universidad Carlos III de Madrid de 8.8 hombres mes (1.155 horas)

EQUIPOS

En este apartado se detallan los gastos relacionados con los equipos informáticos utilizados a lo largo del proyecto.

Descripción	Coste	% Uso dedicado proyecto	Dedicación (meses)	Periodo de depreciación	Coste imputable ^{b)}
Portátil: Acer Aspire 5740	600,00 €	100	4,3	60	343,00 €
Total:					343,00 €

b) Fórmula de cálculo de la Amortización:

$$\frac{A}{B} \times C \times D$$

A = nº de meses desde la fecha de facturación en que el equipo es utilizado

B = periodo de depreciación (60 meses)

C = coste del equipo (sin IVA)

D = % del uso que se dedica al proyecto (habitualmente 100%)

SUBCOBTRATACIÓN DE TAREAS

En este apartado se reflejan las tareas que se han delegado a empresas externas. Como en el desarrollo de este proyecto no se ha llevado a cabo ninguna subcontratación con terceros, los gastos son nulos.

Descripción	Empresa	Coste imputable
-	-	-
Total:		0,00 €

OTROS COSTES DIRECTOS DEL PROYECTO ^{c)}

Descripción	Empresa	Coste imputable
Microsoft Office 2010	Microsoft	120,00 €
Conexión a Internet	Movistar	100,00 €
Suministro eléctrico	Iberdrola	120,00 €
Total:		340,00 €

c) Este capítulo de gastos incluye todos los gastos no contemplados en conceptos anteriores, por ejemplo: fungible, viajes y dietas

6. Resumen de costes

ESTIMACIÓN DE COSTES TOTALES

En este apartado se muestra la estimación de costes totales del proyecto utilizando los cálculos de los apartados anteriores.

Concepto	Coste
Personal	14.864,67 €
Amortización	343,00 €
Subcontratación de tareas	0,00 €
Otros costes directos	340,00 €
Total costes directos	15.547,67 €
Costes indirectos	3.109,53 €
Total gastos sin IVA	18.657,20 €
IVA (18%)	3.358,30 €
Total:	22.015,50 €

7.3.3 PRESUPUESTO PARA EL CLIENTE

En esta sección se detalla el presupuesto a presentar al cliente, en el que se incluyen los gastos vistos en la sección anterior más un coste añadido en concepto de riesgos, para hacer frente a posibles imprevistos durante el desarrollo del proyecto.

También se añadirá al coste final los beneficios esperados por el desarrollo del proyecto, los cuales se calcularán como un porcentaje del coste total incluidos los riesgos.

El porcentaje de riesgos elegido para este proyecto es del 10%, por ser un valor utilizado en proyectos previos.

En cuanto a los beneficios, se ha definido un porcentaje del 15% en concepto de beneficios por el desarrollo del proyecto, por la misma razón que el porcentaje de riesgos.

A continuación se muestra el presupuesto total a entregar al cliente:

PRESUPUESTO DEL CLIENTE

Concepto	Coste
Personal	14.864,67 €
Amortización	343,00 €
Subcontratación de tareas	0,00 €
Otros costes directos	340,00 €
Total costes directos	15.547,67 €
Costes indirectos	3.109,53 €
Total costes sin riesgo	18.657,20 €
Riesgo (10%)	1865,72 €
Total costes sin beneficios	20.522,92 €
Beneficios (15%)	3.078,44 €
Total gastos sin IVA	23.601,36 €
IVA (18%)	4.248,24 €
Total:	27.849,61€

7.3.4 COSTE FINAL Y ANÁLISIS DE DESVIACIÓN

En este apartado se presenta el coste final resultante del desarrollo del proyecto. El coste real se calcula de la misma manera que el estimado, pero utilizando el tiempo real empleado para el desarrollo del proyecto.

También se presenta una comparativa entre el presupuesto estimado inicialmente con el coste final del proyecto, en el que se analiza la desviación producida en la estimación de los costes y los principales motivos que la han provocado.

El presupuesto real del proyecto es el siguiente:



UNIVERSIDAD CARLOS III DE MADRID

Escuela Politécnica Superior

PRESUPUESTO REAL DEL PROYECTO

7. Autor:
Mario Herreros Díaz
8. Departamento:
Informática
9. Descripción del proyecto:
 - Título: Análisis de ofuscación en Apps mediante casos de prueba
 - Duración (meses): 5,7 meses
 - Tasa de costes indirectos: 20%
10. Presupuesto total del proyecto (€): 28.406,85
11. Desglose presupuestario (costes directos)

PERSONAL

En este apartado se detallan los gastos de personal. Para el cálculo de estos gastos se ha tenido en cuenta la participación de una única persona. La dedicación de esta persona es de cuatro horas diarias en días laborables y sábados durante los 171 días reales que ha durado el desarrollo del proyecto.

El coste hombre/mes se ha establecido en 2.694,39 €, que es el valor indicado en la plantilla utilizada como base para un ingeniero. También se ha tenido en cuenta el importe a pagar a la Seguridad Social, el cual es un 28,3% del sueldo bruto del trabajador.

Apellidos y nombre	Categoría	Dedicación ^{a)} Persona/ mes	Coste Persona/ mes	Coste bruto	Seguridad Social	Coste total
Herreros Díaz, Mario	Ingeniero	5,75	2.694,39 €	2.694,39 €	762,51€	3.456,90 €
Total:						19.704,33 €

- a) Hombre mes = 131.25 horas. Máximo anual de dedicación de 12 hombres mes (1.575 horas)
Máximo anual para PDI de la Universidad Carlos III de Madrid de 8.8 hombres mes (1.155 horas)

EQUIPOS

En este apartado se detallan los gastos relacionados con los equipos informáticos utilizados a lo largo del proyecto.

Descripción	Coste	% Uso dedicado proyecto	Dedicación (meses)	Periodo de depreciación	Coste imputable ^{b)}
Portátil: Acer Aspire 5740	600,00 €	100	5,7	60	357,00 €
Total:					357,00 €

b) Fórmula de cálculo de la Amortización:

$$\frac{A}{B} \times C \times D$$

A = nº de meses desde la fecha de facturación en que el equipo es utilizado

B = periodo de depreciación (60 meses)

C = coste del equipo (sin IVA)

D = % del uso que se dedica al proyecto (habitualmente 100%)

SUBCOBTRATACIÓN DE TAREAS

En este apartado se reflejan las tareas que se han delegado a empresas externas. Como en el desarrollo de este proyecto no se ha llevado a cabo ninguna subcontratación con terceros, los gastos son nulos.

Descripción	Empresa	Coste imputable
-	-	-
Total:		0,00 €

OTROS COSTES DIRECTOS DEL PROYECTO^{c)}

Descripción	Empresa	Coste imputable
Microsoft Office 2010	Microsoft	120,00 €
Conexión a Internet	Movistar	100,00 €
Suministro eléctrico	Iberdrola	120,00 €
Total:		340,00 €

c) Este capítulo de gastos incluye todos los gastos no contemplados en conceptos anteriores, por ejemplo: fungible, viajes y dietas.

12. Resumen de costes reales**COSTES TOTALES REALES**

En este apartado se muestra costes totales reales del proyecto utilizando los cálculos de los apartados anteriores.

Concepto	Coste
Personal	19.704,33 €
Amortización	357,00 €
Subcontratación de tareas	0,00 €
Otros costes directos	340,00 €
Total costes directos	20.061,33 €
Costes indirectos	4.012,27 €
Total gastos sin IVA	24.073,60 €
IVA (18%)	4.333,25 €
Total:	28.406,85 €

La Tabla 116 se muestra el coste estimado, el coste real y la variación cometida en la estimación:

Concepto	Coste estimado	Coste real	Variación
Costes directos	15.547,67 €	20.061,33 €	- 4.513,66 €
Costes de personal	14.864,67 €	19.704,33 €	- 4.839,66 €
Costes de equipo	343,00 €	357,00 €	-14,00 €
Otros costes	340,00 €	340,00 €	0,00 €
Costes indirectos	3.109,53 €	4.012,27 €	-902,74 €
Total	18.657,20 €	24.073,60 €	-5.416,40 €

Tabla 116: Desviación del presupuesto.

Como se puede observar, se produce un sobre coste no estimado de 5.416,40 €. Esto es debido principalmente al aumento del tiempo invertido en el proyecto realmente respecto al estimado inicialmente. Invirtiendo los beneficios estimados en este sobre coste, arrojarían unas pérdidas de 2.337,96 €. Y restando estas pérdidas al importe reservado para riesgos se sigue perdiendo 472,24 €.

Este tipo de problemas se pueden solventar aumentando el porcentaje de riesgo aplicados o el porcentaje de beneficios, teniendo así un mayor margen para solucionar problemas en las estimaciones iniciales.

Las principales causas de este resultado negativo han sido la mala planificación inicial realizada y la incorrecta elección de los porcentajes de beneficios y riesgos utilizada en la estimación de costes.

ANEXO I

BIBLIOGRAFÍA

1. **Yajin Zhou, Xuxian Jiang.** Dissecting Android Malware: Characterization and Evolution. *Proceedings of the 33rd IEEE Symposium on Security and Privacy (Oakland 2012)*. San Francisco, CA, USA : s.n., Mayo de 2012.
2. Juniper Networks. *At Risk: Global Mobile Threat Study Finds Security Vulnerabilities at all Time Highs for Mobile Devices*. [En línea] 10 de Mayo de 2011. [Citado el: 5 de Febrero de 2013.] <http://juniper.mwnewsroom.com/manual-releases/2011/At-Risk--Global-Mobile-Threat-Study-Finds-Security>.
3. **Apvrille, Axelle.** Cryptography for mobile malware. *RSA Conference Europe*. Octubre de 2011.
4. **Jiang, Xuxian.** Security Alert: New Sophisticated Android Malware DroidKungFu Found in Alternative Chinese App Markets. *Security Alert: New Sophisticated Android Malware DroidKungFu Found in Alternative Chinese App Markets*. [En línea] [Citado el: 6 de Septiembre de 2013.] <http://www.csc.ncsu.edu/faculty/jiang/DroidKungFu.html>.
5. **Zhou, Yajin y Jiang, Xuxian.** An Analysis of the AnserverBot Trojan. Carolina del Norte : s.n., 2011.
6. **Valenzuela, Ismael.** Últimos avances en Análisis Forense de sistemas Android. Málaga, Andalucía, España : s.n., 17 de Abril de 2012.
7. **Object Management Group.** UML Resource Page. [En línea] [Citado el: 6 de Septiembre de 2013.] <http://www.uml.org/>.
8. **Framingham, Mass.** IDC. *IDC*. [En línea] 14 de Febrero de 2013. [Citado el: 4 de Julio de 2013.] <http://www.idc.com/getdoc.jsp?containerId=prUS23946013>.
9. **Framingham, Mass.** IDC. *IDC*. [En línea] 16 de Mayo de 2013. [Citado el: 7 de Julio de 2013.] <http://www.idc.com/getdoc.jsp?containerId=prUS24108913>.
10. **Mateo, San.** IDC. *IDC*. [En línea] 1 de Mayo de 2013. [Citado el: 4 de Julio de 2013.] <http://www.idc.com/getdoc.jsp?containerId=prUS24093213>.
11. **Brähler, Stefan.** Analysis of the Android Architecture. *Analysis of the Android Architecture*. Karlsruhe : s.n., 2010.
12. **Cordero, Nicolás.** VoCSDroid. *VoCSDroid*. Leganés : s.n., 2012.
13. Android Developers Gingerbread. [En línea] [Citado el: 29 de Julio de 2013.] <http://developer.android.com/about/versions/android-2.3-highlights.html>.
14. Android Developers Honeycomb. [En línea] [Citado el: 29 de Julio de 2013.] <http://developer.android.com/about/versions/android-3.0-highlights.html>.

15. Android Developers Ice Cream Sandwich. [En línea] [Citado el: 29 de Julio de 2013.] <http://developer.android.com/about/versions/android-4.0-highlights.html>.
16. Android Developers Jelly Bean. [En línea] [Citado el: 29 de Julio de 2013.] <http://developer.android.com/about/versions/jelly-bean.html>.
17. **Cabrera, Fredy**. Android Malware. Asunción : s.n., 2012.
18. **Manjunath, Vibha**. Reverse Engineering Of Malware On Android. Essex : s.n., 2011.
19. **You, Ilun y Yim, Kangbin**. Malware Obfuscation Techniques: A Brief Survey. Seúl : s.n., 2010.
20. **Cap, Clemens, Aust, Sebastien y Bräuning, Christine**. Network Security & Forensics. Tartu : s.n., 2012.
21. Android Apktool. *Android Apktool*. [En línea] [Citado el: 5 de Julio de 2013.] <https://code.google.com/p/android-apktool/>.
22. dex2jar. *dex2jar*. [En línea] [Citado el: 5 de Julio de 2013.] <http://code.google.com/p/dex2jar/>.
23. Java Decompiler. *Java Decompiler*. [En línea] [Citado el: 5 de Julio de 2013.] <http://java.decompiler.free.fr/?q=jdgui>.
24. smali. *smali*. [En línea] [Citado el: 5 de Julio de 2013.] <https://code.google.com/p/smali/>.
25. AntiLVL - Android License Verification Library Subversion. [En línea] [Citado el: 6 de Julio de 2013.] http://androidcracking.blogspot.com.es/p/antilvl_01.html.
26. Radare, the reverse engineering framework . [En línea] [Citado el: 6 de Julio de 2013.] <http://www.radare.org/y/>.
27. DroidBox. Android Application Sandbox. [En línea] [Citado el: 6 de Julio de 2013.] <http://code.google.com/p/droidbox/>.
28. DECAF Binary Analysis Platform. [En línea] [Citado el: 6 de Julio de 2013.] <http://code.google.com/p/decaf-platform/wiki/DroidScope>.
29. **Kwong Yan, Lok y Yin, Heng**. DroidScope: Seamlessly Reconstructing the OS and Dalvik Semantic Views. Syracuse : s.n., 2012.
30. TCPDUMP & LIBCAP. *TCPDUMP & LIBCAP*. [En línea] [Citado el: 11 de Agosto de 2013.] <http://www.tcpdump.org/>.
31. Android Developers Monkeyrunner. [En línea] [Citado el: 11 de Agosto de 2013.] http://developer.android.com/tools/help/monkeyrunner_concepts.html.

32. **Milano, Diego.** GitHub: AndroidViewClient. [En línea] [Citado el: 11 de Agosto de 2013.] <https://github.com/dtmilano/AndroidViewClient>.

33. Robotium. [En línea] [Citado el: 11 de Agosto de 2013.] <http://code.google.com/p/robotium/>.

34. **Jiang, Xuxian.** Security Alert: New Sophisticated Android Malware DroidKungFu Found in Alternative Chinese App Markets . *Department of Computer Science, NC State University*. [En línea] 23 de Junio de 2011. [Citado el: 8 de Julio de 2013.] <http://www.csc.ncsu.edu/faculty/jiang/DroidKungFu.html>.

35. **Shannon, Claude E.** A Mathematical Theory of Communication. *Bell System Technical Journal*. 1948, Vol. 27.

ANEXO II

EVALUACIÓN: RESULTADOS DE COMPORTAMIENTO

En este apartado se presentan de manera introductoria los datos obtenidos en los resultados referentes al comportamiento de los casos de prueba realizados por el sistema respecto al comportamiento ordinario de las aplicaciones. Los datos que se presentan son los que son utilizados por el sistema para determinar si los elementos analizados esconden una funcionalidad ofuscada de carácter malicioso.

Las principales operaciones que el sistema utiliza para llevar a cabo su detección se definen en los siguientes grupos:

- ***L/E***: agrupa el total de operaciones de lectura y escritura en disco realizadas.
- ***Net***: reúne todas las operaciones de red detectadas durante la ejecución del caso de prueba, siendo estas conexiones abiertas, escrituras en red y lecturas en red.
- ***Leaks***: representa el total de fugas de información producidas.
- ***SMS***: indica el total de intentos de envío de mensajes SMS producidos.
- ***Calls***: agrupa todas las llamadas realizadas durante la ejecución.
- ***DNS***: expresa el total de peticiones DNS producidas durante el caso de prueba con el fin de identificar posible actividad con botnets.
- ***Net cifrada***: agrupa el total de direcciones de peticiones DNS realizadas y contenido de peticiones HTTP GET y HTTP POST que presentan ofuscación. En algunos casos el *malware* envía información cifrada a servidores externos, para que la descifren y utilicen estos últimos.
- ***Net no cifrada***: reúne el total de direcciones de peticiones DNS producidas cuya dirección esté en claro, más el número de peticiones HTTP POST y GET cuyo contenido también se haya envía en texto plano.
- ***Op cripto***: representa el total de todas las operaciones criptográficas, cifrados y descifrados de datos principalmente, realizadas durante los análisis dinámicos.

Por cada grupo se detallarán los siguientes resultados:

- **Comportamiento ordinario**: se presentará los resultados obtenidos en los análisis de comportamiento de las aplicaciones sin modificación. Se presenta la media de los resultados cosechados en los dos análisis dinámicos realizados por cada aplicación. La gráfica que se presenta en este apartado ordena los valores resultantes de mayor a menor, para facilitar el entendimiento del total de operaciones realizadas de cada tipo.
- **Condicionales**: se explicará el comportamiento obtenido en los casos de prueba centrados en analizar las sentencias condicionales. Se presentará una gráfica en la que se presenta el comportamiento en cada una de las ejecuciones: mutación de condiciones a la inversa, condiciones verdaderas, condiciones falsas y condiciones aleatorias.
- **Assets**: se detallará el comportamiento en los casos de prueba centrados en la mutación de ficheros *assets* frente a los obtenidos en las ejecuciones iniciales.

- **API criptográfica:** se presentará los comportamientos obtenidos en los casos de prueba cuyo fin era alterar las llamadas a APIs criptográficas, relacionándolos con el comportamiento corriente de la aplicación.
- **Cadenas ofuscadas:** se indicará las variaciones detectadas tras la realización de los casos de prueba relacionados con la mutación de cadenas de texto ofuscadas.

Se expondrá un análisis del tipo Condicionales, Assets, API criptográfica y Cadenas ofuscadas por cada uno de los grupos explicados anteriormente.

Se han dividido las aplicaciones analizadas en dos grupos:

- Grupo I (G1): Este grupo está formado por 100 aplicaciones, de las cuales 22 de ellas pertenecen a la familia ADRD y el resto a la familia AnserverBot.
- Grupo II (G2): Este grupo está formado por las aplicaciones restantes de la familia AnserverBot, DroidKungFu, Pjapps, y GingerMaster.

COMPORTAMIENTO ORDINARIO

En esta sección se presenta el tipo de operaciones ocurridas durante la ejecución de las aplicaciones originales utilizadas en este estudio.

En cada una de las dos gráficas se presenta el nivel de operaciones de cada tipo producidas en cada una de las aplicaciones, ordenadas de mayor a menor número de operaciones.

En la Ilustración 58 se muestra el comportamiento normal del conjunto de aplicaciones analizadas en el G1:

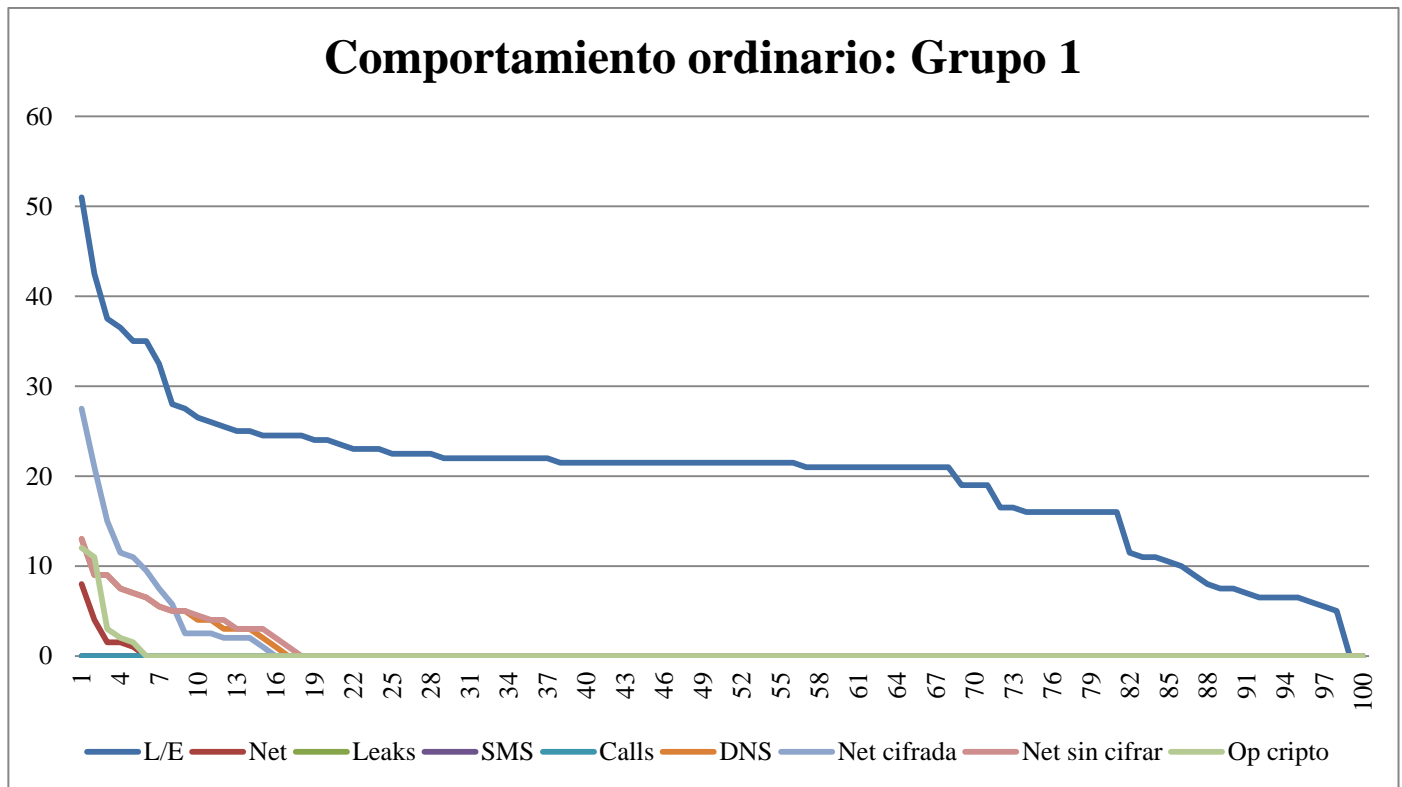


Ilustración 58: Comportamiento ordinario: Grupo 1.

En el Grupo 1 se han producido operaciones de *L/E* en casi toda la totalidad de las aplicaciones, operaciones *DNS* en no más de una veintena, al igual que las operaciones del tipo *Net cifrada* y *Net claro*. En cuanto a operaciones criptográficas y operaciones de tipo *Net*, se ha producido cierta actividad en no más de siete aplicaciones analizadas. Del resto de los tipos (*SMS*, *Calls* y *Leaks*) no se han producido actividades.

En la Ilustración 59 se muestra el comportamiento normal del conjunto de aplicaciones analizadas en el G2:

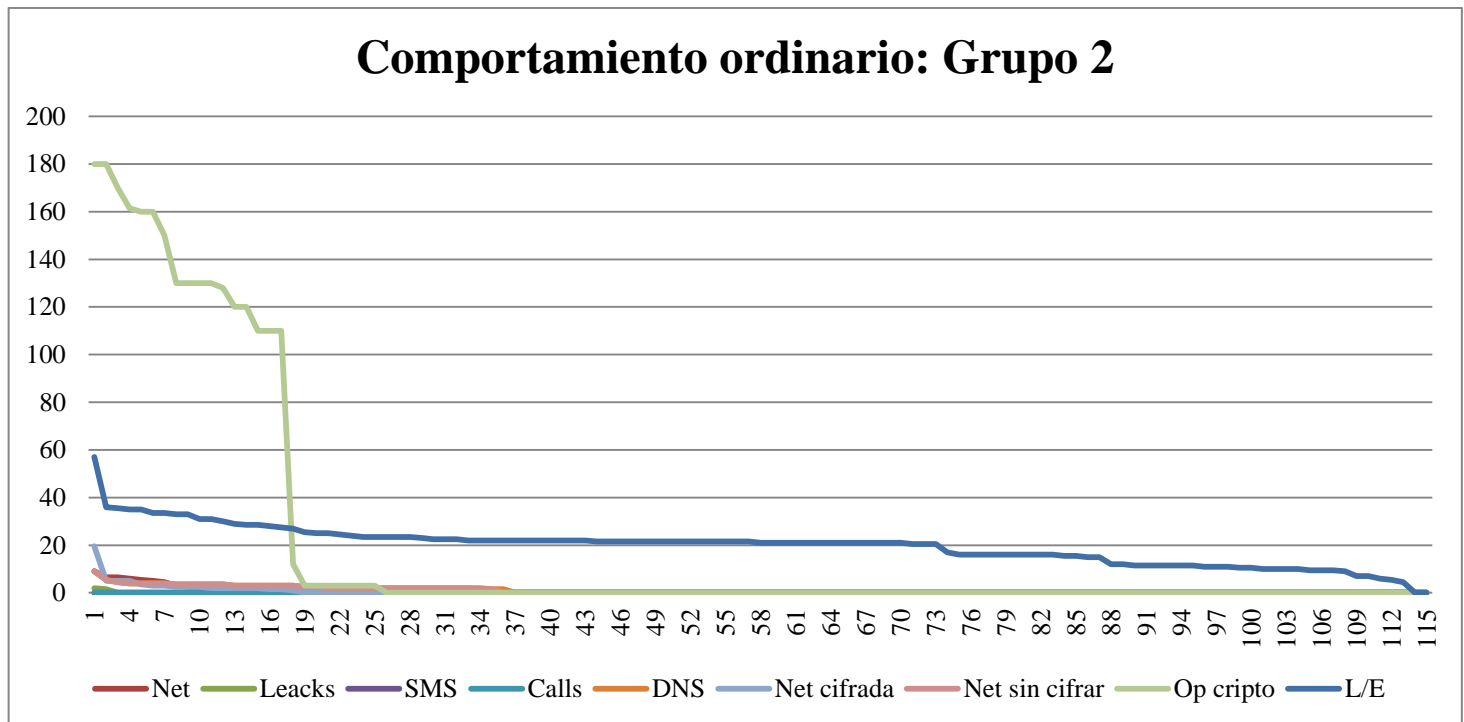


Ilustración 59: Comportamiento ordinario: Grupo 2.

En el Grupo 2 se han producido operaciones de *L/E* en casi toda la totalidad de las aplicaciones, operaciones *DNS* en unas 35 aplicaciones, al igual que las operaciones de red. En cuanto a operaciones criptográficas, en 25 aplicaciones se han producido eventos de este tipo. En dos aplicaciones se han producido operaciones del tipo *Leacks*, mientras que no se ha producido ninguna operación del tipo *SMS* ni *Calls*.

Se observa que hay un alto porcentaje de aplicaciones que únicamente presenta operaciones de lectura y escritura. Por lo que en este tipo de aplicaciones la detección de elementos ofuscados maliciosos resulta una labor difícil. En menor proporción, hay aplicaciones que presentan actividad de red y operaciones criptográficas. En dos de las aplicaciones se han detectado fugas de información en sus correspondientes ejecuciones. Mientras, de los tipos *SMS* y *Calls* no se ha producido ninguna operación.

ASSETS

Este apartado se centra en el comportamiento resultante obtenido en los casos de prueba centrados en ficheros *assets*.

- **ASSETS: L/E**

En esta sección se presentan los niveles de operaciones del tipo *L/E* producidos en casos de prueba del tipo *Assets*.

Las gráficas presentan el porcentaje de actividad *L/E* producidas en la ejecución de la aplicación sin modificar (Ordinario) y en cada una de las ejecuciones de los artefactos *assets* analizados (En esta caso cada aplicación puede presentar de 1 a n ejecuciones,

dependiendo del número de ficheros *assets* analizados). El eje de ordenadas presenta el porcentaje de actividad producida en cada ejecución y el eje de abscisas presenta la aplicación en la que se ha producido dicho resultado.

En la Ilustración 60 se presenta los resultados obtenidos al modificar los ficheros *assets* presentes en las aplicaciones del Grupo 1:

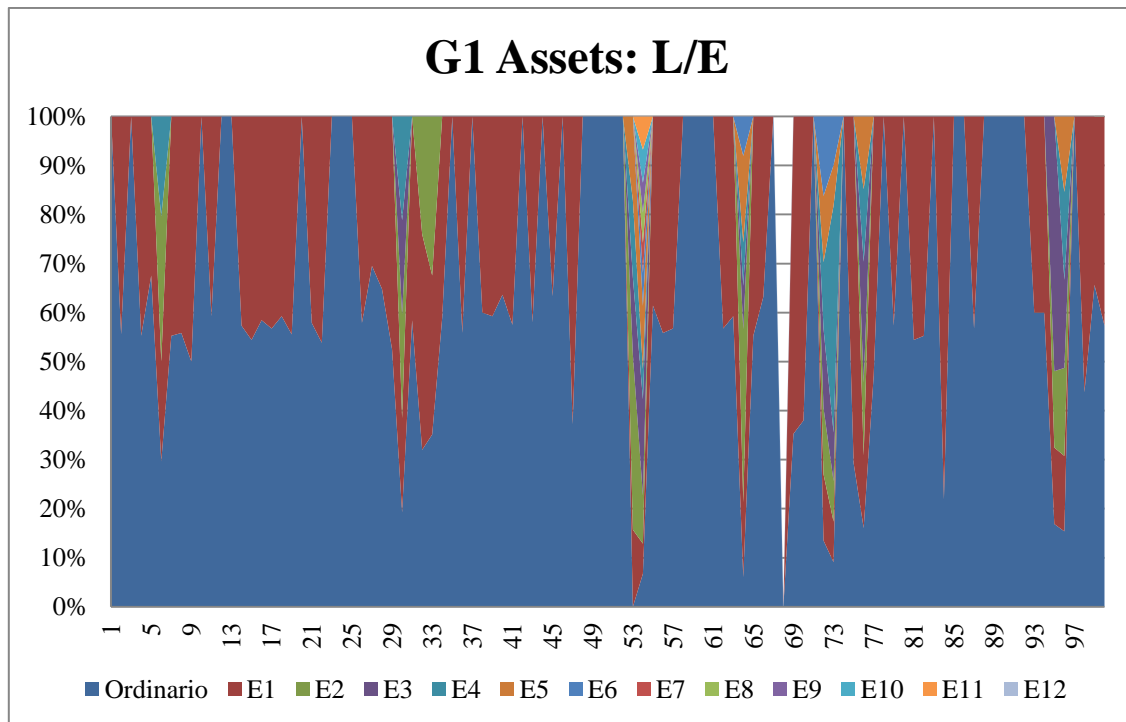


Ilustración 60: G1 Assets: L/E.

Las principales particularidades ocurridas en este grupo son las siguientes:

- El 94% de las aplicaciones analizadas presentan al menos un fichero en la carpeta *assets*,
- En un total de 30 ejecuciones se han producido resultados nulos respecto al caso ordinario.
- En la mayoría de los casos ha disminuido significativamente el número de operaciones respecto al caso ordinario.

En la Ilustración 61 se presenta los resultados obtenidos al modificar los ficheros *assets* presentes en las aplicaciones del Grupo 2:

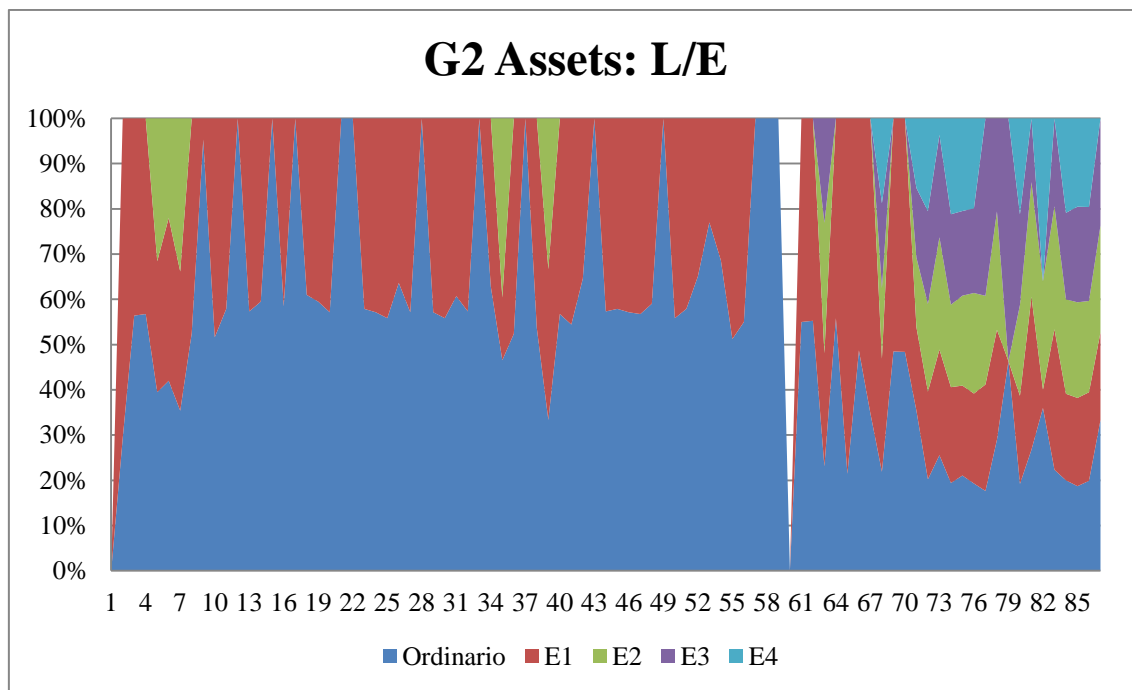


Ilustración 61: G2 Assets: L/E.

Las principales particularidades ocurridas en este grupo son las siguientes:

- Un 76% de las aplicaciones de este grupo presentan, al menos, un fichero en su directorio *assets*.
- En un 20% de las aplicaciones, alguna de las ejecuciones presentan una fuerte variación respecto a los niveles de operación de la ejecución ordinaria.

Los datos obtenidos muestran que hay ficheros *assets* que al ser modificados, afectan al porcentaje ordinario de operaciones de lectura y escritura de la funcionalidad habitual de algunas aplicaciones analizadas.

• ASSETS: NET

Se han descartado todos los resultados que hayan sido nulos tanto en el caso ordinario como en las ejecuciones de los casos de prueba. También se han eliminado los resultados de las aplicaciones que no poseían ficheros *assets*.

En esta sección se presentan los niveles de operaciones del tipo *Net* producidos en casos de prueba del tipo Assets.

Las gráficas presentan el porcentaje de actividad *Net* producidas en la ejecución de la aplicación sin modificar (Ordinario) y en cada una de las ejecuciones de los artefactos *assets* analizados (En esta caso cada aplicación puede presentar de 1 a n ejecuciones, dependiendo del número de ficheros *assets*). El eje de ordenadas presenta el porcentaje de actividad producida en cada ejecución y el eje de abscisas presenta la aplicación en la

que se ha producido dicho resultado. Los niveles negativos indican que para dicha aplicación no se ha producido el número de ejecución indicado por la leyenda.

Únicamente se han obtenido resultados de tres aplicaciones para el Grupo 1, cuyo comportamiento se muestra en la Ilustración 62:

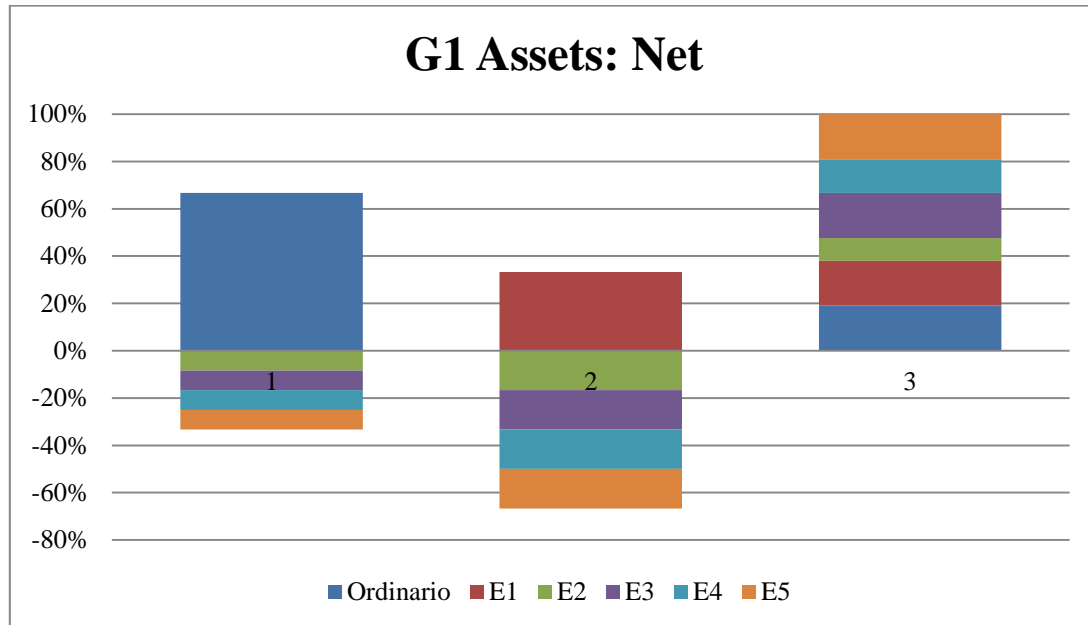


Ilustración 62: G1 Assets: Net

Los rasgos destacables son los siguientes:

- La aplicación 1 con 8 operaciones tipo *Net*, en su única ejecución (E1) se han obtenido resultados nulos.
- La aplicación 3 ha mantenido un porcentaje de operaciones similar al caso ordinario en sus cinco ejecuciones de caso de prueba.

Para el Grupo 2 se han obtenido los siguientes resultados mostrados en la Ilustración 63:

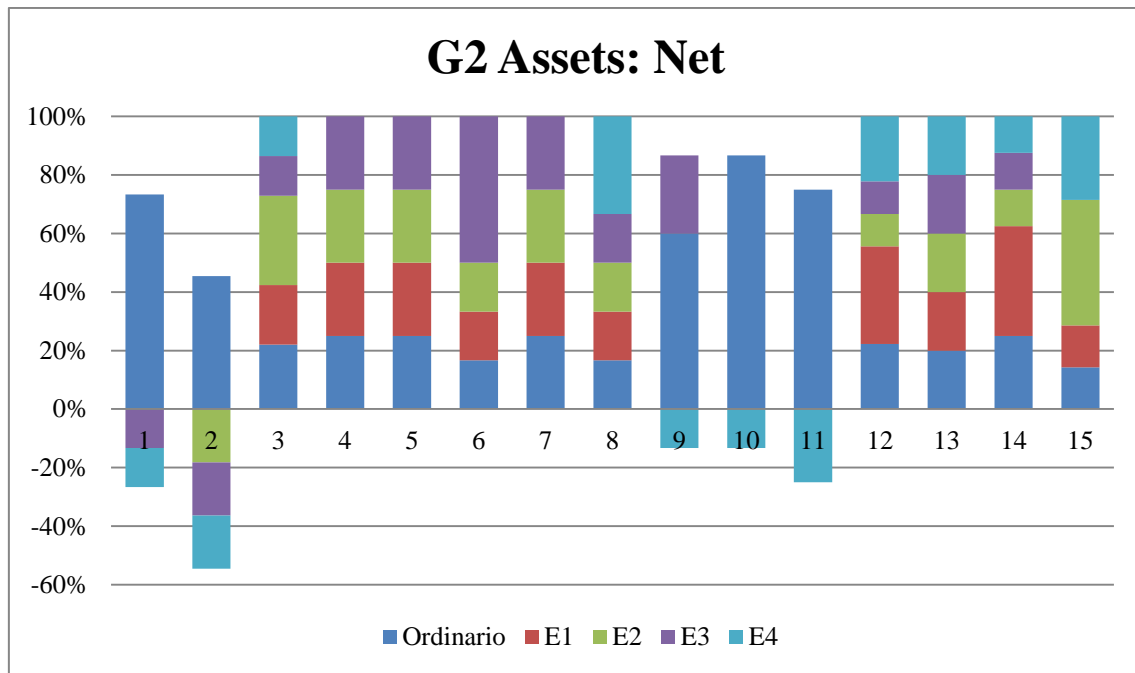


Ilustración 63: G2 Assets: Net.

En un 60% de las aplicaciones que presentaban actividad de red han presentado valores nulos o muy inferiores respecto a la ejecución ordinaria respecto a este tipo de operación.

Con los resultados obtenidos se puede observar que la modificación de ciertos ficheros *assets* puede alterar el comportamiento normal de este tipo de operaciones.

- **ASSETS: LEAKS**

Las aplicaciones que presentan este tipo de operación en su comportamiento ordinario, pertenecen al Grupo 2, concretamente a la familia DroidKungFu. En todas las ejecuciones en los que los ficheros *assets* los resultados obtenidos en este tipo de operación han sido nulos.

- **ASSETS: SMS**

En los resultados obtenidos de este grupo de aplicaciones no se han producido operaciones del tipo SMS.

- **ASSETS: CALLS**

En los resultados obtenidos de este grupo de aplicaciones no se han producido operaciones del tipo Calls.

• ASSETS: DNS

Se han desechado aquellas aplicaciones cuyos resultados en las pruebas iniciales y en los casos de prueba sean nulos.

En esta sección se presentan los niveles de operaciones del tipo *DNS* producidos en casos de prueba del tipo Assets.

Las gráficas presentan el porcentaje de actividad *DNS* producidas en la ejecución de la aplicación sin modificar (Ordinario) y en cada una de las ejecuciones de los artefactos *assets* analizados (En esta caso cada aplicación puede presentar de 1 a n ejecuciones, dependiendo del número de ficheros *assets*). El eje de ordenadas presenta el porcentaje de actividad producida en cada ejecución y el eje de abscisas presenta la aplicación en la que se ha producido dicho resultado. Los resultados negativos representan que para dicha aplicación no se ha producido dicho número de ejecución.

En la Ilustración 64 se muestran los resultados obtenidos para el Grupo 1:

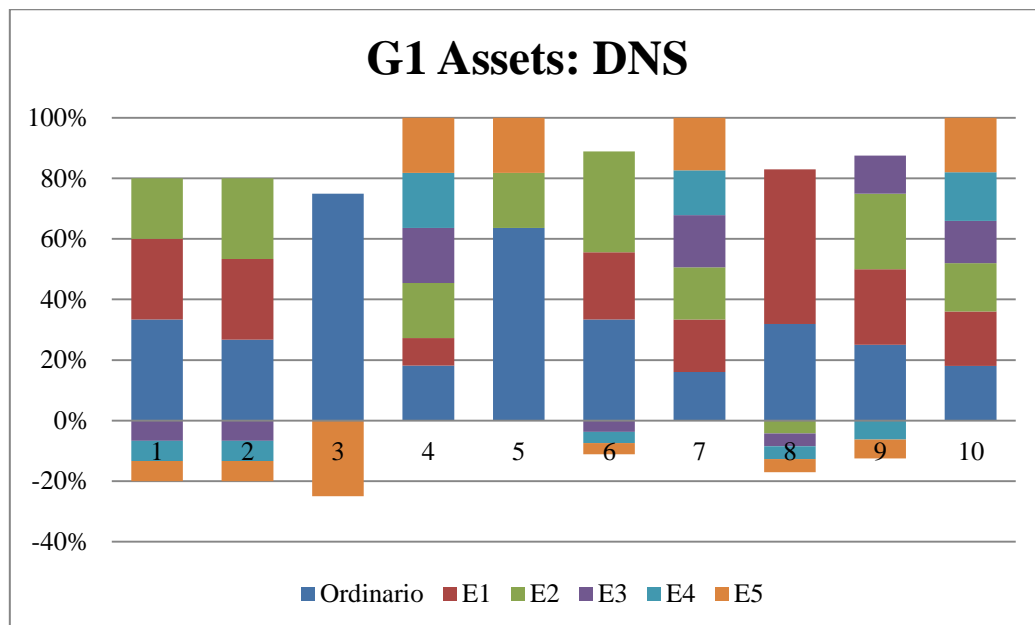


Ilustración 64: G1 Assets: DNS.

. Las principales características a destacar son las siguientes:

- En la aplicación 3 se obtienen resultados nulos en las cuatro ejecuciones de caso de prueba, mientras que en el caso ordinario se produjo actividad del tipo DNS.
- En la aplicación 5 se han obtenido resultados nulos en las ejecuciones E1, E3 y E4, y en el resto ha disminuido en más de la mitad la actividad DNS respecto al caso ordinario.
- El resto de aplicaciones han obtenido niveles de actividad similares al caso ordinario en sus distintas ejecuciones.

En la Ilustración 65 se muestran los resultados obtenidos para el Grupo 2:

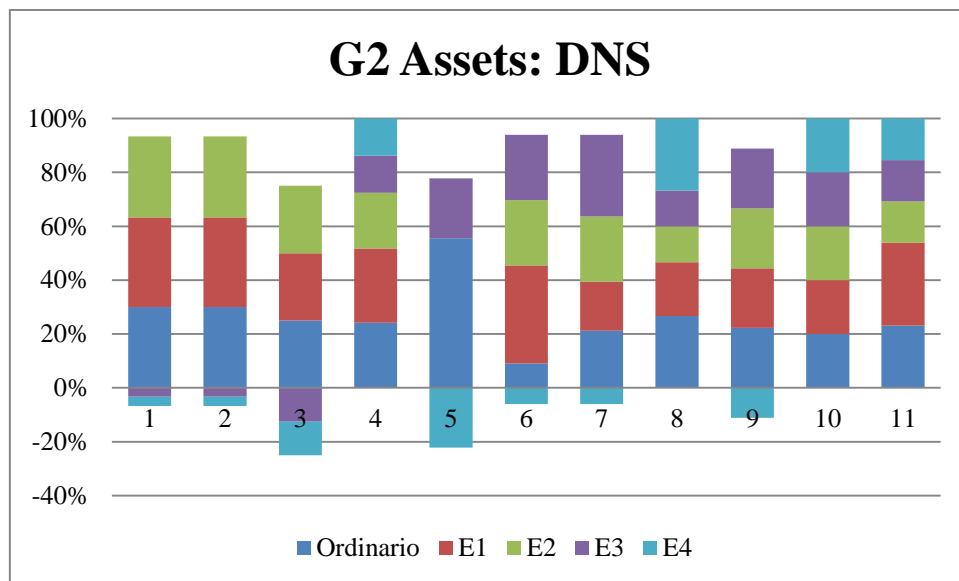


Ilustración 65: G2 Assets: DNS.

Para este grupo, el 20% presenta fuerte variación respecto al comportamiento ordinario de las aplicaciones.

Esta información se resume en que un porcentaje de los ficheros *assets* presentan una funcionalidad vinculada a este tipo de operación.

- **ASSETS: NET CIFRADO**

Al igual que en casos anteriores, se han descartado aquellos resultados nulos. Se han estudiado un total de diez aplicaciones para el Grupo 1 y 9 aplicaciones para el Grupo 2.

En esta sección se presentan los niveles de operaciones del tipo *Net cifrado* producidos en casos de prueba del tipo Assets.

Las gráficas presentan el porcentaje de actividad *Net cifrado* producidas en la ejecución de la aplicación sin modificar (Ordinario) y en cada una de las ejecuciones de los artefactos *assets* analizados (En esta caso cada aplicación puede presentar de 1 a n ejecuciones, dependiendo del número de ficheros *assets*). El eje de ordenadas presenta el porcentaje de actividad producida en cada ejecución y el eje de abscisas presenta la aplicación en la que se ha producido dicho resultado. Los niveles negativos indican que para dicha aplicación no se ha producido el número de ejecución indicado por la leyenda.

En la Ilustración 66 se muestra la actividad producida en la ejecución de los distintos casos de prueba:

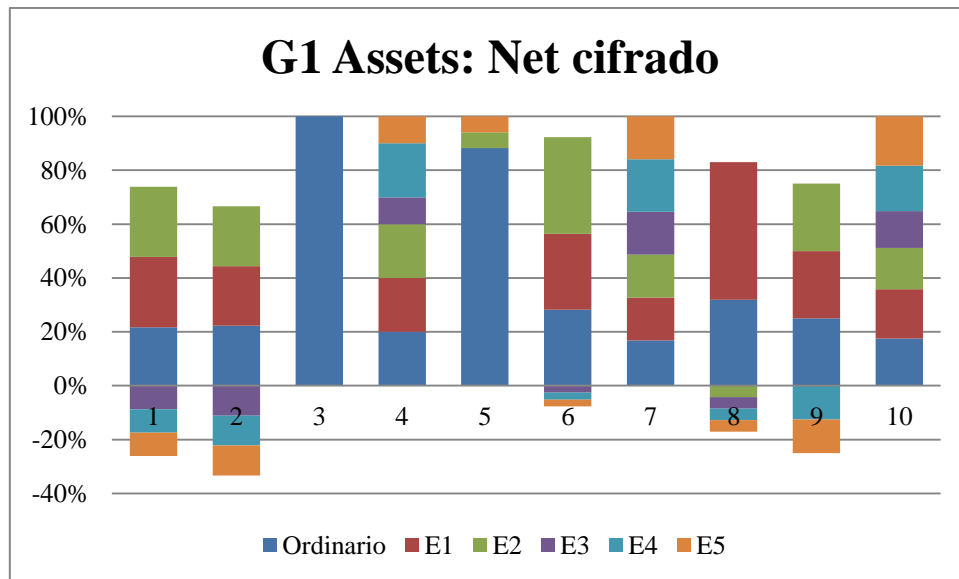


Ilustración 66: G1 Assets: Net cifrado.

Los resultados negativos indican que dicho número de ejecución no fue ejecutado en dicha aplicación, por tener menor número de ficheros *assets*. Los resultados indican:

- La aplicación 3 y 5, presentan resultados nulos en la mayoría de sus ejecuciones de caso de prueba, y en la aplicación 9 en su ejecución E3.
- El resto de aplicaciones presenta una actividad similar al del caso ordinario.

En la Ilustración 67 se muestra el comportamiento obtenido por las aplicaciones del Grupo 2:

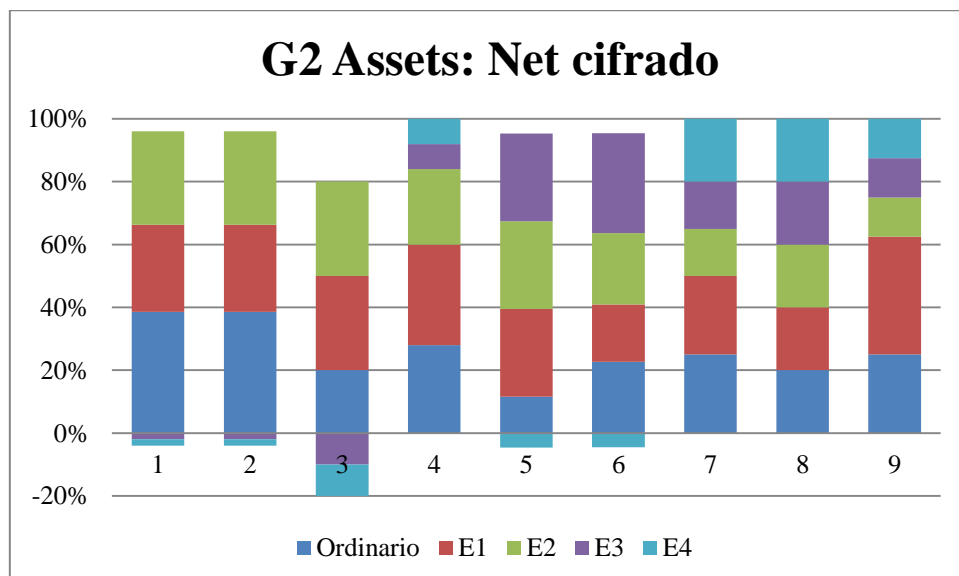


Ilustración 67: G2 Assets: Net cifrado.

En esta ocasión, no se han obtenido resultados que varíen notoriamente respecto al comportamiento ordinario de la aplicación.

Para las operaciones de *Net cifrado* se ha encontrado cierta variación en un pequeño porcentaje de ficheros *assets*, pero menor que respecto a otros tipos de operación

- **ASSETS: NET SIN CIFRAR**

Eliminando los resultados nulos y las aplicaciones que no presentan este directorio, se han analizado diez aplicaciones que presentan actividad de este tipo tanto para el Grupo 1 como para el Grupo 2.

En esta sección se presentan los niveles de operaciones del tipo *Net sin cifrar* producidos en casos de prueba del tipo Assets.

Las gráficas presentan el porcentaje de actividad *Net sin cifrar* producidas en la ejecución de la aplicación sin modificar (Ordinario) y en cada una de las ejecuciones de los artefactos *assets* analizados (En esta caso cada aplicación puede presentar de 1 a n ejecuciones, dependiendo del número de ficheros *assets*). El eje de ordenadas presenta el porcentaje de actividad producida en cada ejecución y el eje de abscisas presenta la aplicación en la que se ha producido dicho resultado. Los niveles negativos indican que para dicha aplicación no se ha producido el número de ejecución indicado por la leyenda.

En la Ilustración 68 se muestra los resultados obtenidos:

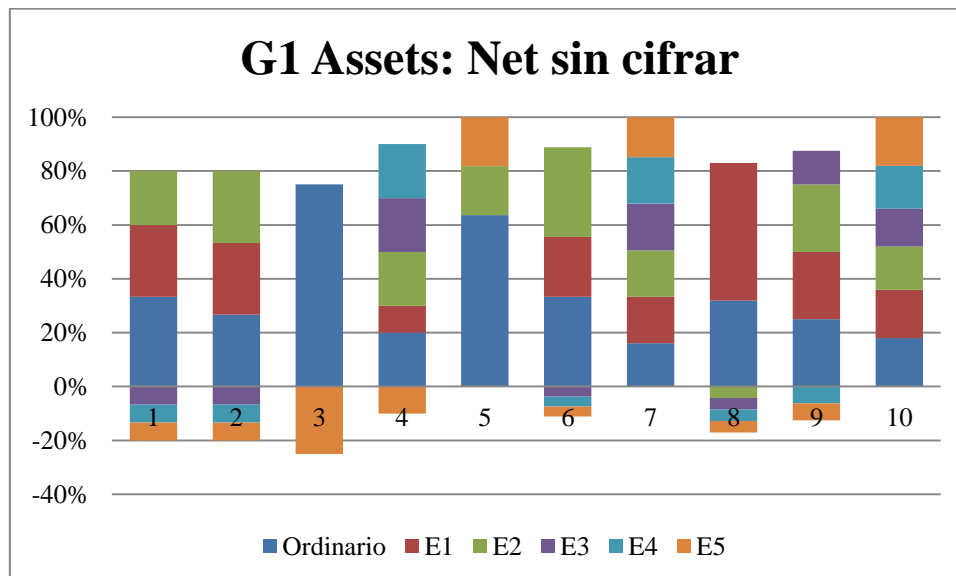


Ilustración 68: G1 Assets: Net sin cifrar.

Los principales rasgos presentes en estos resultados son:

- La aplicación 3 y 5 presentan resultados nulos en casi todas sus ejecuciones, mientras que en su comportamiento ordinario presentaban actividad de este tipo.
- El resto de aplicaciones presentan una actividad similar al comportamiento ordinario.

En la Ilustración 69 se muestra el comportamiento resultante para las aplicaciones del Grupo 2:

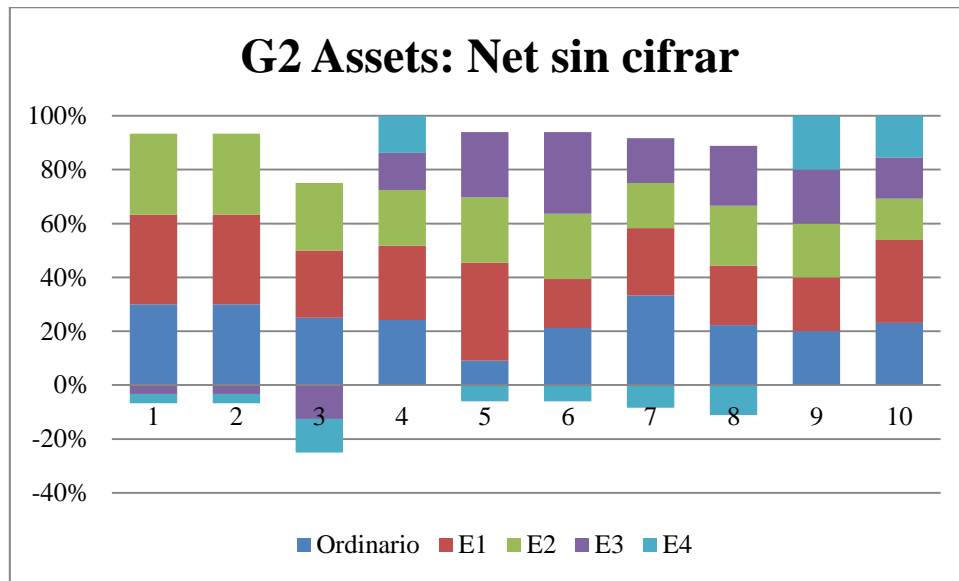


Ilustración 69: G2 Assets: Net sin cifrar.

El comportamiento de cada una de las ejecuciones de caso de prueba presenta niveles similares al del caso ordinario para cada una de las aplicaciones analizadas.

Como resultado global, destacar que el comportamiento resultante recogido de los casos de prueba es similar a los recogidos durante la ejecución ordinaria.

- **ASSETS: OP CRIPTO**

No se han producido casos de prueba del tipo *assets* cuyos resultados ordinarios y resultados en las ejecuciones sean distintos de cero.

CADENAS OFUSCADAS

Este apartado se centra en el comportamiento resultante obtenido en los casos de prueba centrados en cadenas de texto ofuscadas.

- **CADENAS OFUSCADAS: L/E**

En esta sección se presentan los niveles de operaciones del tipo *L/E* producidos en casos de prueba del tipo Cadenas ofuscadas.

Las gráficas presentan el porcentaje de actividad *L/E* sin cifrar producidas en la ejecución de la aplicación sin modificar (Ordinario) y en cada una de las ejecuciones de las cadenas de texto analizadas (En esta caso cada aplicación puede presentar de 1 a n ejecuciones, dependiendo del número de cadenas ofuscadas analizadas). El eje de ordenadas presenta el porcentaje de actividad producida en cada ejecución y el eje de

abscisas presenta la aplicación en la que se ha producido dicho resultado. Los niveles negativos indican que para dicha aplicación no se ha producido el número de ejecución indicado por la leyenda.

En la Ilustración 70 se muestra la gráfica que representa los niveles de actividad del tipo L/E, tanto en el caso ordinario como en las distintas ejecuciones de los casos de prueba del tipo Cadena ofuscada para las aplicaciones pertenecientes al Grupo 1:

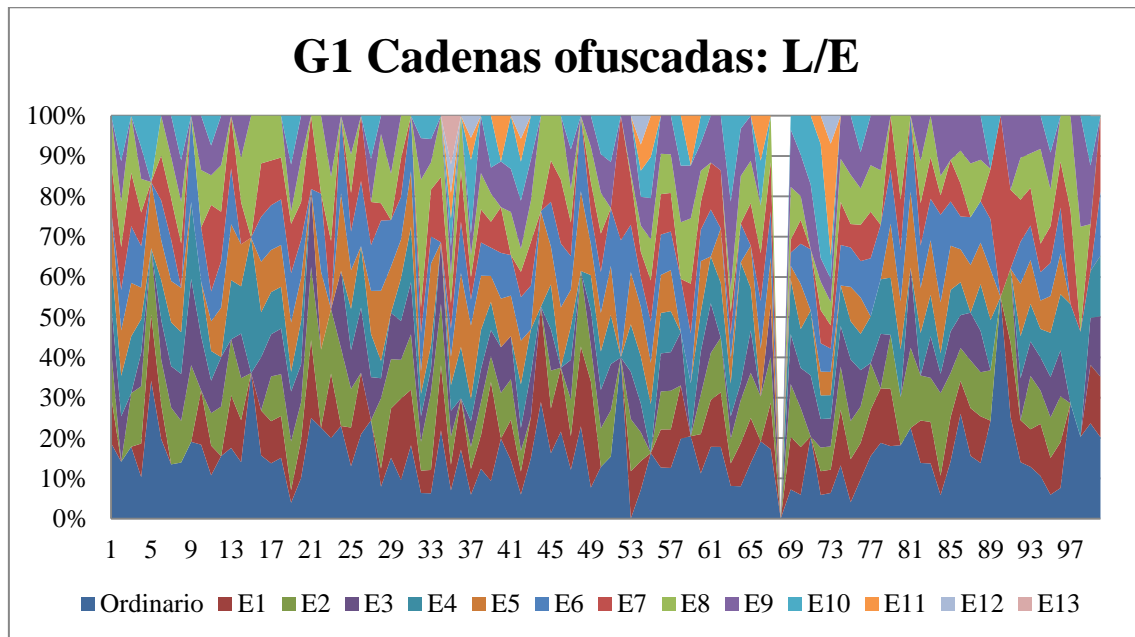


Ilustración 70: G1 Cadenas ofuscadas: L/E.

Los principales rasgos a destacar son los siguientes:

- En la ejecución E1, 25 aplicaciones presentan valores nulos, en la E2 lo presentan 23 aplicaciones, en la E3 unas 23, en la E4 unas 22, en la E5 unas 24, en la E6 unas 17, en la E7 unas 21, en la E8 unas 24, en la E9 unas 25, en la E10 unas 11 y en la E11 en una sola aplicación.
- El resto de ejecuciones presenta niveles de actividad similares al comportamiento ordinario.

En la Ilustración 71, se presenta el comportamiento resultante para el Grupo 2:

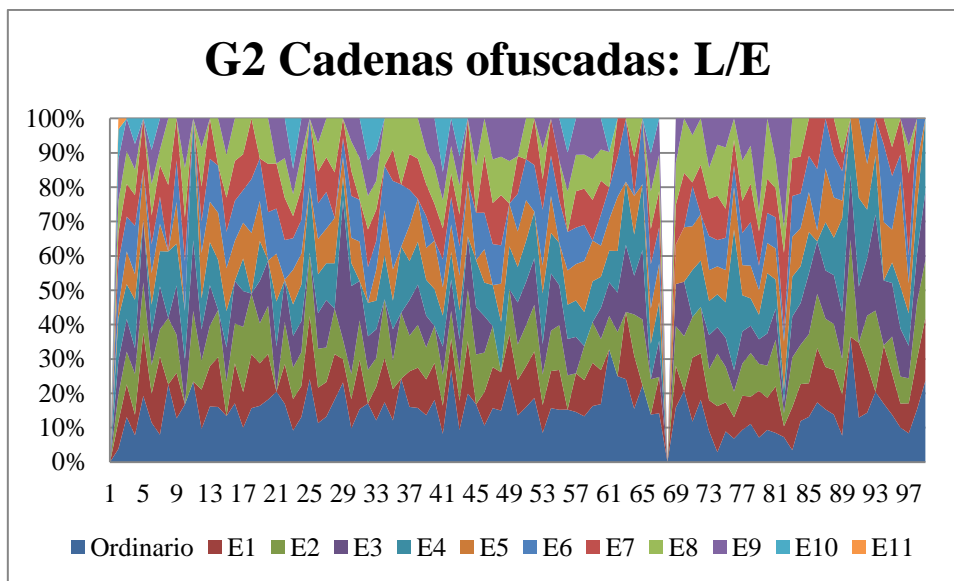


Ilustración 71: G2 Cadenas ofuscadas: L/E.

Aproximadamente, en un 40% de las aplicaciones analizadas existe alguna ejecución en la que se ha modificado una variable ofuscada, consiguiendo una variación notoria del comportamiento ordinario de la aplicación.

• CADENAS OFUSCADAS: NET

Se han descartado todos aquellos resultados cuya actividad del tipo *Net* fuese nula tanto en el comportamiento ordinario como en los casos de prueba.

En esta sección se presentan los niveles de operaciones del tipo *Net* producidos en casos de prueba del tipo Cadenas ofuscadas.

Las gráficas presentan el porcentaje de actividad *Net* sin cifrar producidas en la ejecución de la aplicación sin modificar (Ordinario) y en cada una de las ejecuciones de las cadenas de texto analizadas (En esta caso cada aplicación puede presentar de 1 a n ejecuciones, dependiendo del número de cadenas ofuscadas analizadas). El eje de ordenadas presenta el porcentaje de actividad producida en cada ejecución y el eje de abscisas presenta la aplicación en la que se ha producido dicho resultado. Los niveles negativos indican que para dicha aplicación no se ha producido el número de ejecución indicado por la leyenda.

En la Ilustración 72 se presenta el comportamiento de las aplicaciones del Grupo 1 analizadas:

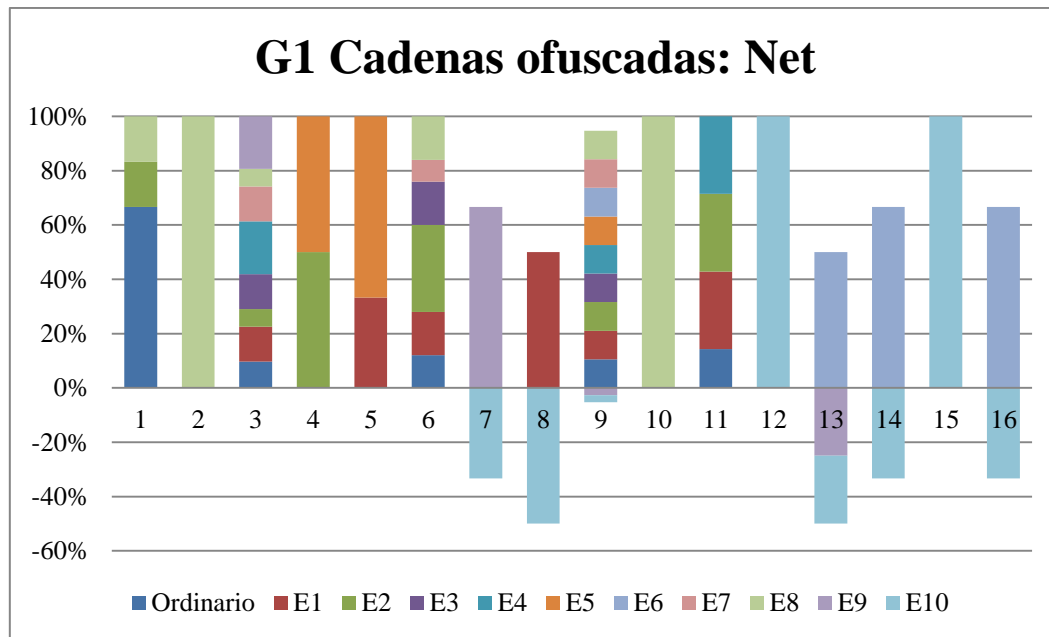


Ilustración 72: G1 Cadenas ofuscadas: Net.

Los niveles negativos indican que para dicha aplicación no se produjo dicho número de ejecución.

Las principales peculiaridades presentes en estos resultados son:

- La aplicación 1 presenta en 8 de las ejecuciones resultados nulos, y en el resto disminuye la actividad en un 75% respecto al original.
- La aplicación 3 y 6 presentan resultados nulos en algunas de sus ejecuciones.
- El resto de ejecuciones presentan niveles similares al del comportamiento ordinario.

Los resultados de comportamiento obtenidos para las aplicaciones del Grupo 2 se observan en la Ilustración 73:

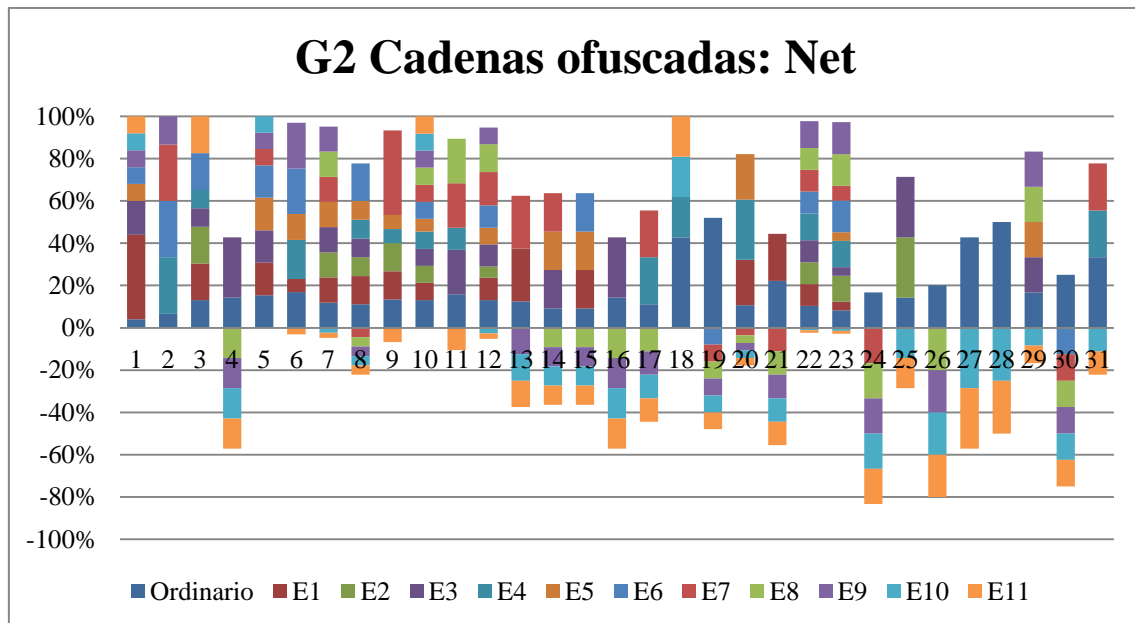


Ilustración 73: G2 Cadenas ofuscadas: Net.

Los datos muestran que en torno al 75% de las aplicaciones han visto variado su comportamiento en cuanto a operaciones de este tipo, en alguna de las variables mutadas.

Esta información concluye que un gran porcentaje de artefactos de este tipo esconden tras de sí funcionalidad relativa a la red.

- **CADENAS OFUSCADAS: LEAKS**

De las aplicaciones del Grupo 2 que presentan este tipo de actividad, en una el 33% de las ejecuciones de caso de prueba han variado notoriamente el comportamiento respecto a la ejecución ordinaria, y en la otra aplicación un 42% de las ejecuciones realizadas.

- **CADENAS OFUSCADAS: SMS**

En los resultados obtenidos de este grupo de aplicaciones no se han producido operaciones del tipo SMS.

- **CADENAS OFUSCADAS: CALLS**

En los resultados obtenidos de este grupo de aplicaciones no se han producido operaciones del tipo Calls.

- **CADENAS OFUSCADAS: DNS**

Se han descartado todos aquellos resultados cuya actividad del tipo *DNS* fuese nula tanto en el comportamiento ordinario como en los casos de prueba.

En esta sección se presentan los niveles de operaciones del tipo *DNS* producidos en casos de prueba del tipo Cadenas ofuscadas.

Las gráficas presentan el porcentaje de actividad *DNS* sin cifrar producidas en la ejecución de la aplicación sin modificar (Ordinario) y en cada una de las ejecuciones de las cadenas de texto analizadas (En esta caso cada aplicación puede presentar de 1 a n ejecuciones, dependiendo del número de cadenas ofuscadas analizadas). El eje de ordenadas presenta el porcentaje de actividad producida en cada ejecución y el eje de abscisas presenta la aplicación en la que se ha producido dicho resultado. Los niveles negativos indican que para dicha aplicación no se ha producido el número de ejecución indicado por la leyenda.

En la Ilustración 74 se presenta los resultados de las aplicaciones analizadas para el Grupo 1:

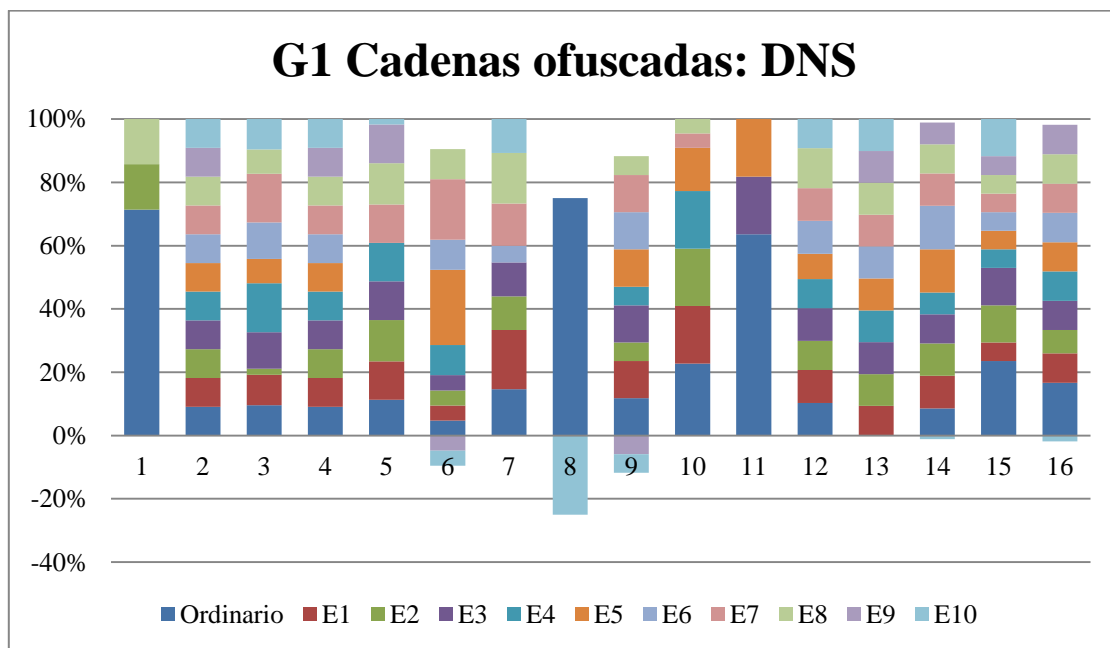


Ilustración 74: G1 Cadenas ofuscadas: DNS.

Aproximadamente, el 57% de las aplicaciones presentan fuertes variaciones en alguna de sus ejecuciones respecto al comportamiento ordinario.

Para el Grupo 2 el comportamiento resultante se presenta en la Ilustración 75:

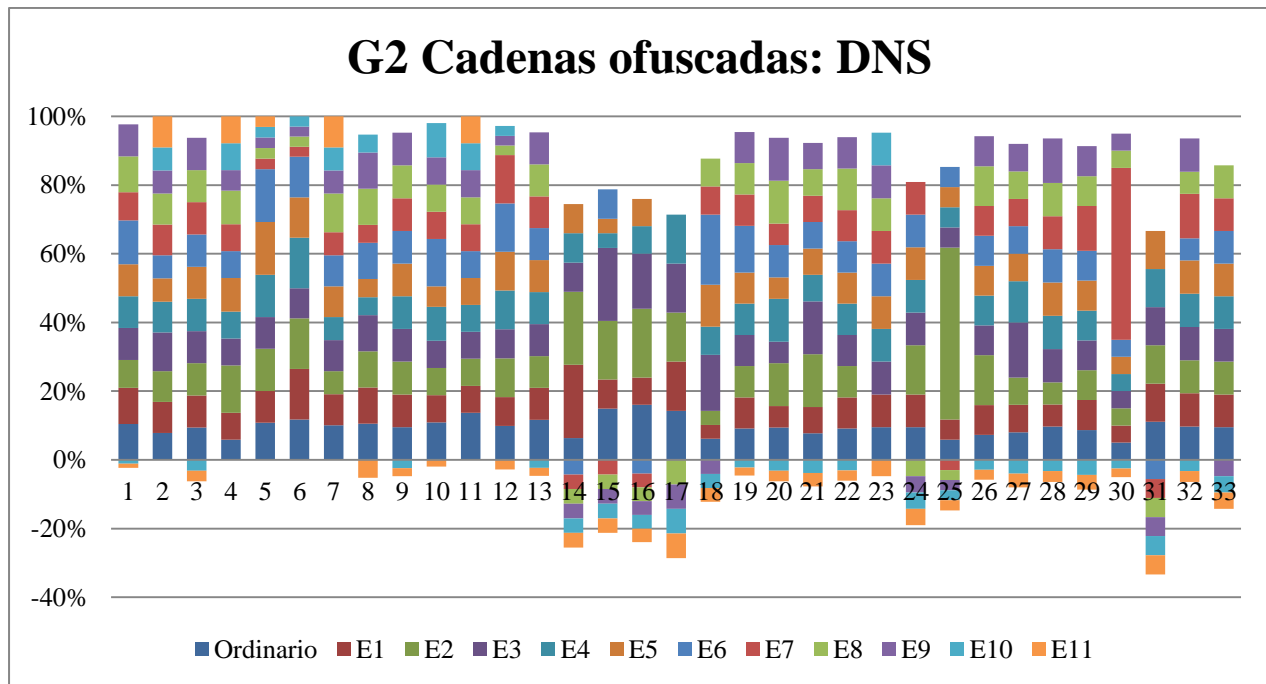


Ilustración 75: G2 Cadenas ofuscadas: DNS.

Se observa que en un 25% de las aplicaciones se produce alguna ejecución en la que una cadena de texto modificada ha arrojado resultados notoriamente diferentes a la ejecución ordinaria.

Con todos estos resultados, se puede concluir que existen cadenas de texto ofuscadas que intervienen directamente en el porcentaje de actividad de este tipo producido en una aplicación.

• CADENAS OFUSCADAS: NET CIFRADO

Se han descartado todos aquellos resultados cuya actividad del tipo *Net cifrado* fuese nula tanto en el comportamiento ordinario como en los casos de prueba.

En esta sección se presentan los niveles de operaciones del tipo *Net cifrado* producidos en casos de prueba del tipo Cadenas ofuscadas.

Las gráficas presentan el porcentaje de actividad *Net cifrado* sin cifrar producidas en la ejecución de la aplicación sin modificar (Ordinario) y en cada una de las ejecuciones de las cadenas de texto analizadas (En esta caso cada aplicación puede presentar de 1 a n ejecuciones, dependiendo del número de cadenas ofuscadas analizadas). El eje de ordenadas presenta el porcentaje de actividad producida en cada ejecución y el eje de abscisas presenta la aplicación en la que se ha producido dicho resultado. Los niveles negativos indican que para dicha aplicación no se ha producido el número de ejecución indicado por la leyenda.

En la Ilustración 76 se presenta el comportamiento de las aplicaciones del Grupo 1 analizadas:

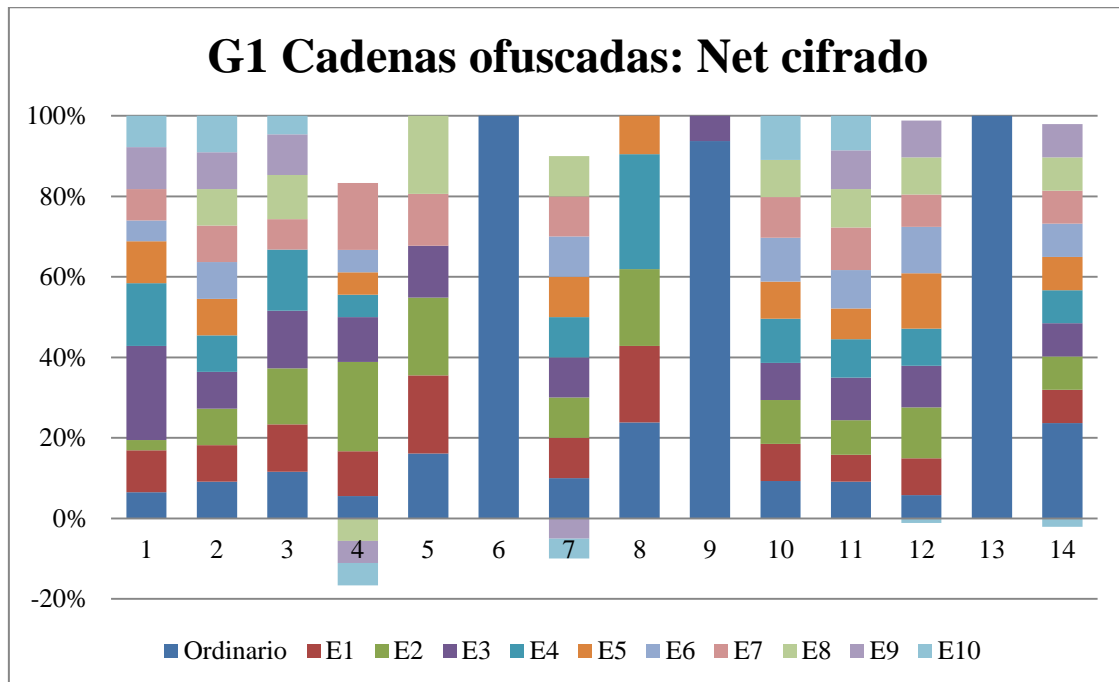


Ilustración 76: G1 Cadenas ofuscadas: Net cifrado.

Los principales rasgos a destacar son:

- Las aplicaciones 1, 8 y 11 presentan una variación total del nivel de actividad DNS en las ejecuciones de casi la totalidad de los casos de prueba.
- Las aplicaciones 3, 5, 7, 10 y 12 en alguna de las ejecuciones se han obtenido resultados nulos.
- El resto de aplicaciones han presentado un comportamiento similar al ordinario en todas sus ejecuciones.

La Ilustración 77 muestra el comportamiento de las aplicaciones del Grupo 2 para este tipo de caso de prueba y tipo de operación:

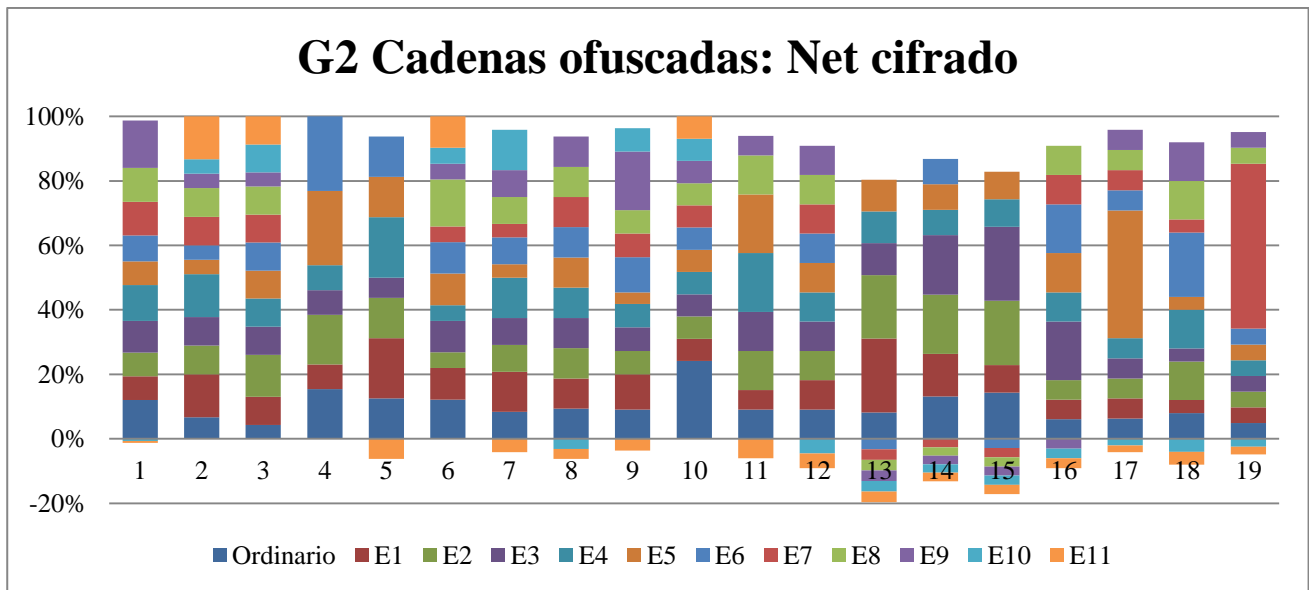


Ilustración 77: G2 Cadenas ofuscadas: Net cifrado.

Se han detectado que en un 32% de las aplicaciones que presentan este tipo de operación, en algunas de sus ejecuciones se han obtenidos resultados muy dispares relativos al comportamiento ordinario.

- **CADENAS OFUSCADAS: NET SIN CIFRAR**

Se han descartado todos aquellos resultados cuya actividad del tipo *Net sin cifrar* fuese nula tanto en el comportamiento ordinario como en los casos de prueba.

En esta sección se presentan los niveles de operaciones del tipo *Net sin cifrar* producidos en casos de prueba del tipo Cadenas ofuscadas.

Las gráficas presentan el porcentaje de actividad *Net sin cifrar* producidas en la ejecución de la aplicación sin modificar (Ordinario) y en cada una de las ejecuciones de las cadenas de texto analizadas (En esta caso cada aplicación puede presentar de 1 a n ejecuciones, dependiendo del número de cadenas ofuscadas analizadas). El eje de ordenadas presenta el porcentaje de actividad producida en cada ejecución y el eje de abscisas presenta la aplicación en la que se ha producido dicho resultado. Los niveles negativos indican que para dicha aplicación no se ha producido el número de ejecución indicado por la leyenda.

En la Ilustración 78 se presenta los resultados de las aplicaciones del Grupo 1 analizadas:

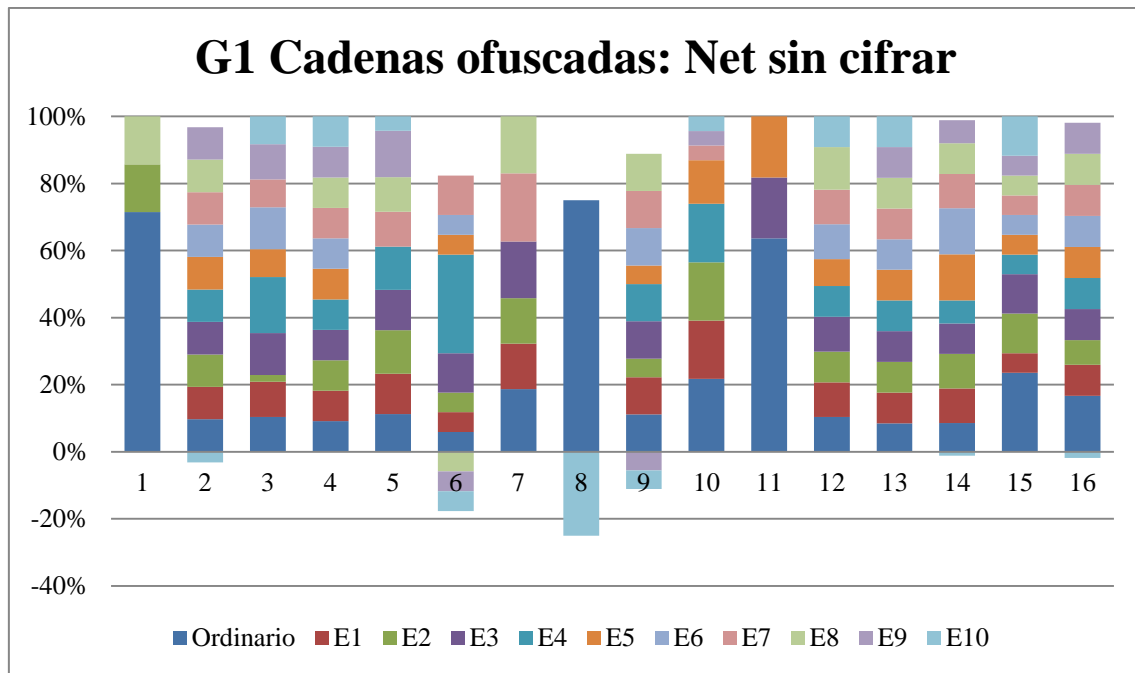


Ilustración 78: G1 Cadenas ofuscadas: Net sin cifrar.

Los principales rasgos recogidos son:

- Existen 3 aplicaciones que presentan valores nulos en el 80% de las ejecuciones.
- Una aplicación presenta valores nulos en el 50% de las ejecuciones.
- Otras 4 aplicaciones presentan valores nulos en torno al 20% de las ejecuciones.
- El resto de aplicaciones presentan valores similares al comportamiento ordinario

En la Ilustración 79 se muestra los mismos tipos de resultados, pero para las aplicaciones del Grupo 2:

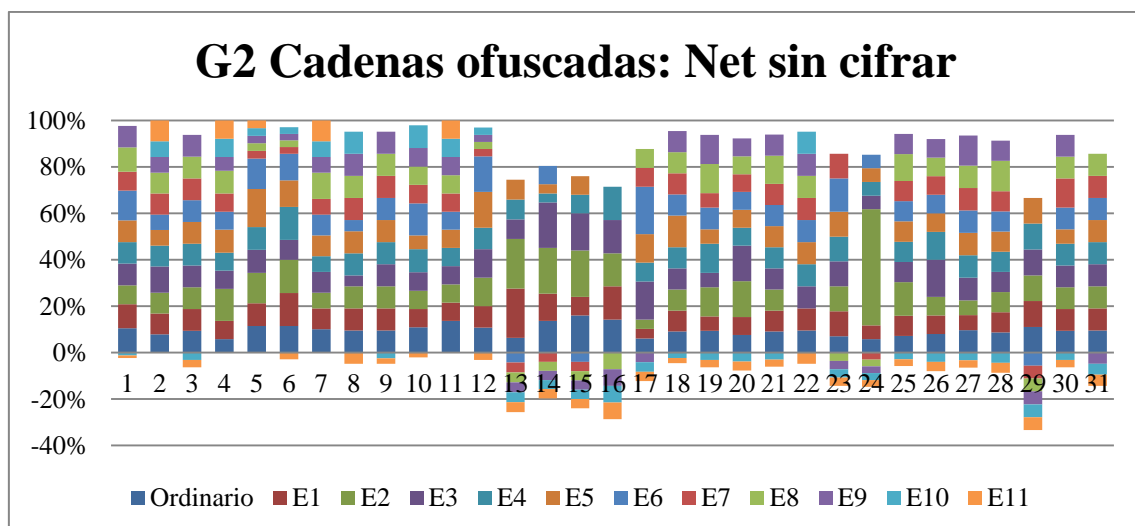


Ilustración 79: G2 Cadenas ofuscadas: Net sin cifrar.

Aproximadamente, el 25% de las aplicaciones presentan una importante variabilidad en alguna de sus ejecuciones de prueba respecto al comportamiento ordinario en este tipo de operación.

• CADENAS OFUSCADAS: OP CRIPTO

En esta sección se presentan los niveles de operaciones del tipo *Op cripto* producidos en casos de prueba del tipo Cadenas ofuscadas.

La gráfica de la Ilustración 80 presenta los niveles de actividad del tipo *Op cripto* en cada una de las ejecuciones por cada aplicación, es decir el eje de las ordenadas presenta el porcentaje de actividad de cada aplicación, mientras que el eje de abscisas presenta el número de ejecución en que se obtuvo dicho resultado. La gráfica de la Ilustración 81 presenta esta información pero a la inversa.

En cinco aplicaciones del Grupo 1 analizadas se han producido resultados distintos a cero tanto en el comportamiento ordinario como en los casos de prueba de modificaciones de strings:

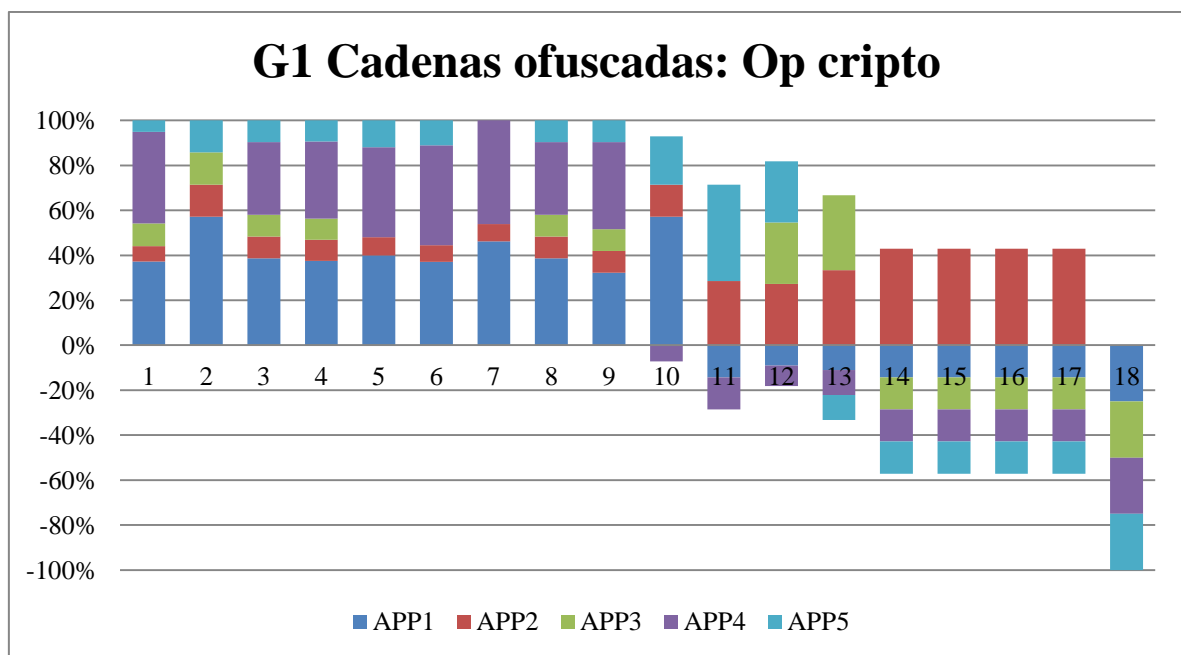


Ilustración 80: G1 Cadenas ofuscadas: Op cripto.

Los principales comportamientos a destacar son:

- La aplicación APP3 presenta 5 ejecuciones en el que los resultados son nulos.
- Las aplicaciones APP2 y APP5 presentan una ejecución con resultado nulo.
- El resto de ejecuciones en las distintas aplicaciones presentan un nivel de actividad criptográfica similar a al comportamiento ordinario.

Los resultados de las aplicaciones del Grupo 2 se presentan en la Ilustración 81:

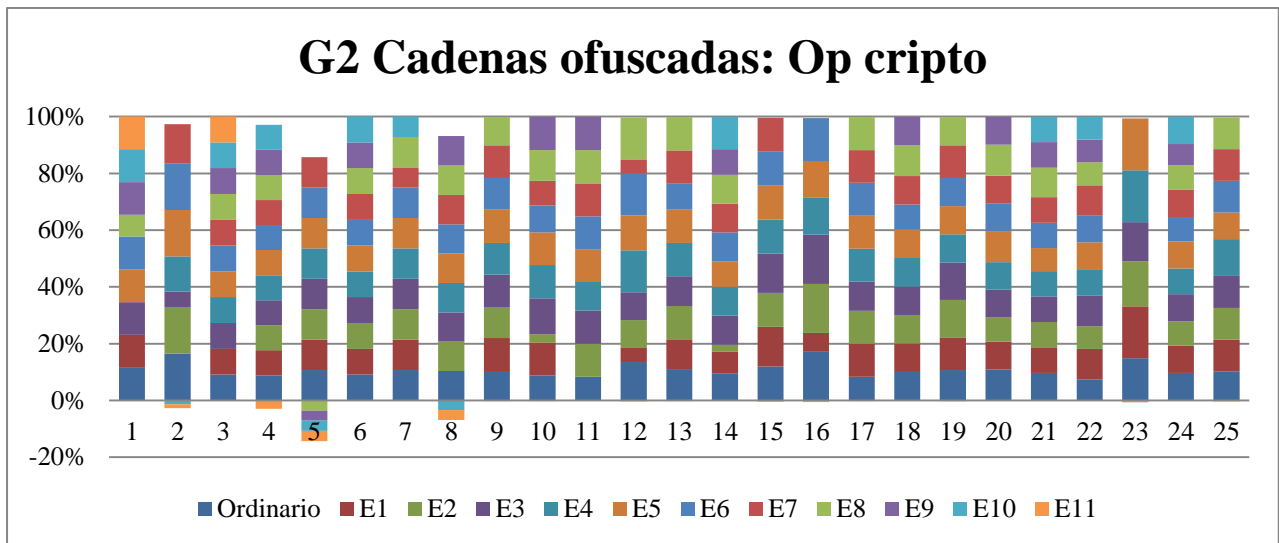


Ilustración 81: G2 Cadenas ofuscadas: Op cripto.

En un 56% de las aplicaciones analizadas del Grupo 2, se muestra una notoria variabilidad en alguna de sus ejecuciones respecto al comportamiento ordinario en cuanto a la actividad criptográfica.

CONDICIONALES

Este apartado se centra en el comportamiento resultante obtenido en los casos de prueba resultantes de modificar las sentencias condicionales del código fuente de la aplicación.

- **CONDICIONALES: L/E**

En esta sección se presentan los niveles de operaciones del tipo *L/E* producidos en casos de prueba del tipo Condicionales.

Las gráficas presentan el porcentaje de actividad en lectura y escritura en disco producidas en las ejecuciones de la aplicación sin modificar (Ordinario) y en las producidas en los casos de prueba basados en invertir las sentencias condicionales presentes en el código (Inversa). También se presentan los porcentajes de *L/E* de los casos de prueba cuyas sentencias condicionales son ciertas en su totalidad (True), de aquellos casos de prueba en que las sentencias condicionales son todas falsas (False) y casos de prueba que las sentencias condicionales han sido modificadas de manera aleatoria (Aleatorio). El eje de ordenadas presenta el porcentaje de actividad producida en cada tipo de caso de prueba y el eje de abscisas presenta la aplicación en la que se ha producido dicho resultado.

En la Ilustración 82 se presenta el comportamiento de cada uno de los casos de prueba de este tipo comparados con el comportamiento normal de las aplicaciones del Grupo 1:

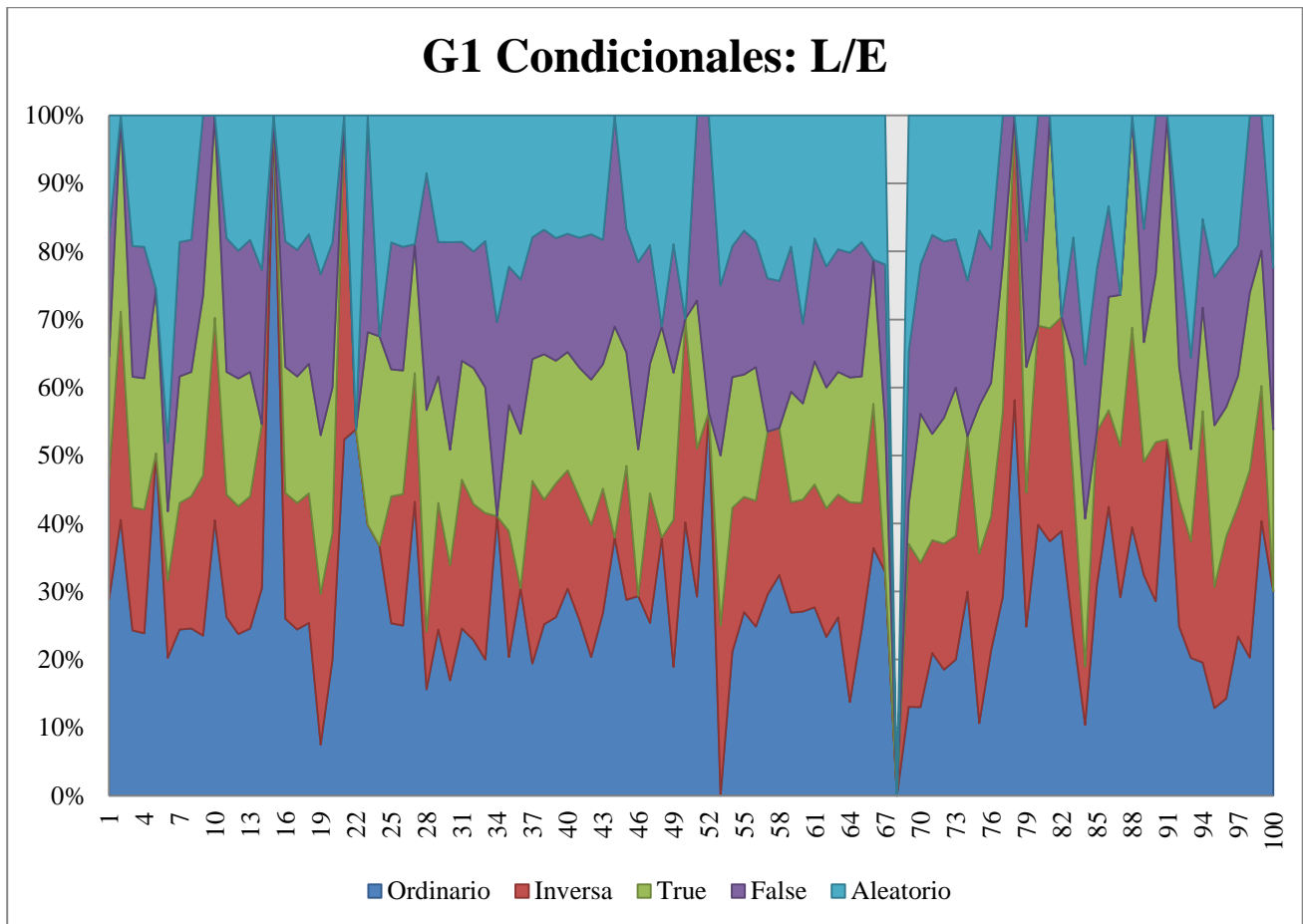


Ilustración 82: G1 Condicionales: L/E.

A niveles generales el número de operaciones de lectura y escritura en disco han disminuido por cada uno de los tipos de casos de prueba respecto al comportamiento ordinario. El total de las operaciones producidas en ejecuciones sin modificación presentan aproximadamente el 30%, mientras que las ejecuciones del tipo Inversa, True y False representan aproximadamente un 10% cada una, y las ejecuciones del tipo Aleatorio representan cerca de un 20% del total.

En la Ilustración 83 se muestra el mismo tipo de información pero para las aplicaciones del Grupo 2:

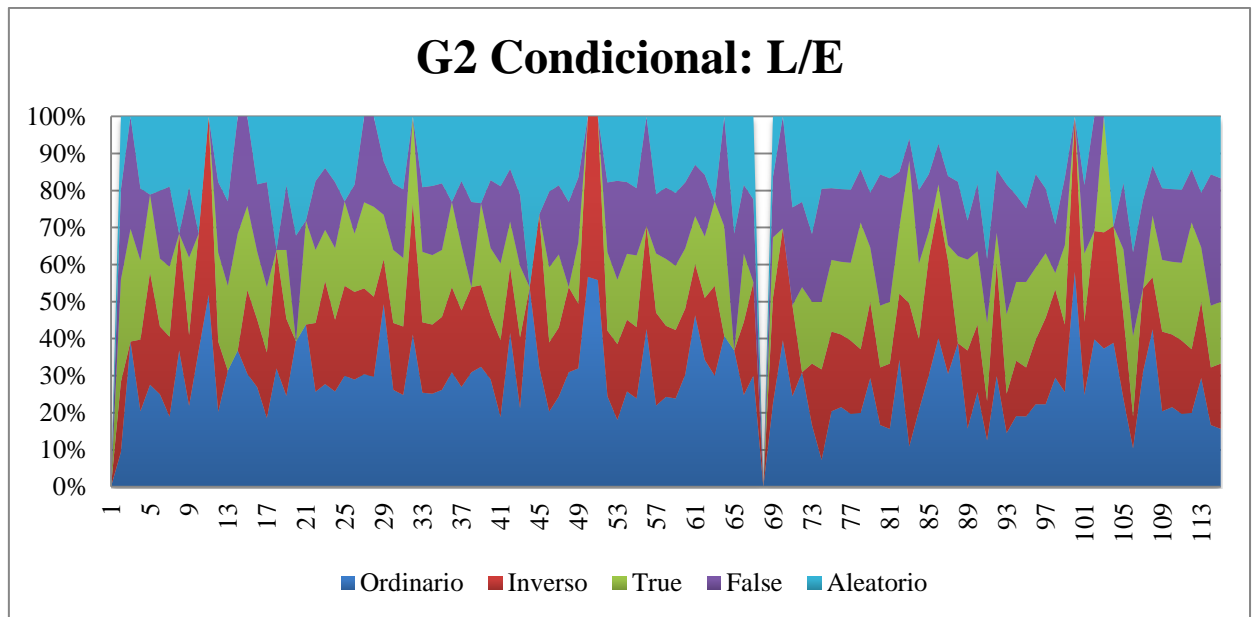


Ilustración 83: G2 Condicionales: L/E.

- **CONDICIONALES: NET**

Se ha descartado todos los casos en que tanto en el comportamiento ordinario como los casos de prueba relacionados con las sentencias condicionales los resultados obtenidos hayan sido nulos.

En esta sección se presentan los niveles de operaciones del tipo *Net* producidos en casos de prueba del tipo Condicionales.

Las gráficas presentan el porcentaje de red producidas en las ejecuciones de la aplicación sin modificar (Ordinario) y en las producidas en los casos de prueba basados en invertir las sentencias condicionales presentes en el código (Inversa). También se presentan los porcentajes de *Net* de los casos de prueba cuyas sentencias condicionales son ciertas en su totalidad (True), de aquellos casos de prueba en que las sentencias condicionales son todas falsas (False) y casos de prueba que las sentencias condicionales han sido modificadas de manera aleatoria (Aleatorio). El eje de ordenadas presenta el porcentaje de actividad producida en cada tipo de caso de prueba y el eje de abscisas presenta la aplicación en la que se ha producido dicho resultado.

En la Ilustración 84 se muestra el comportamiento resultante de las aplicaciones del Grupo 1:

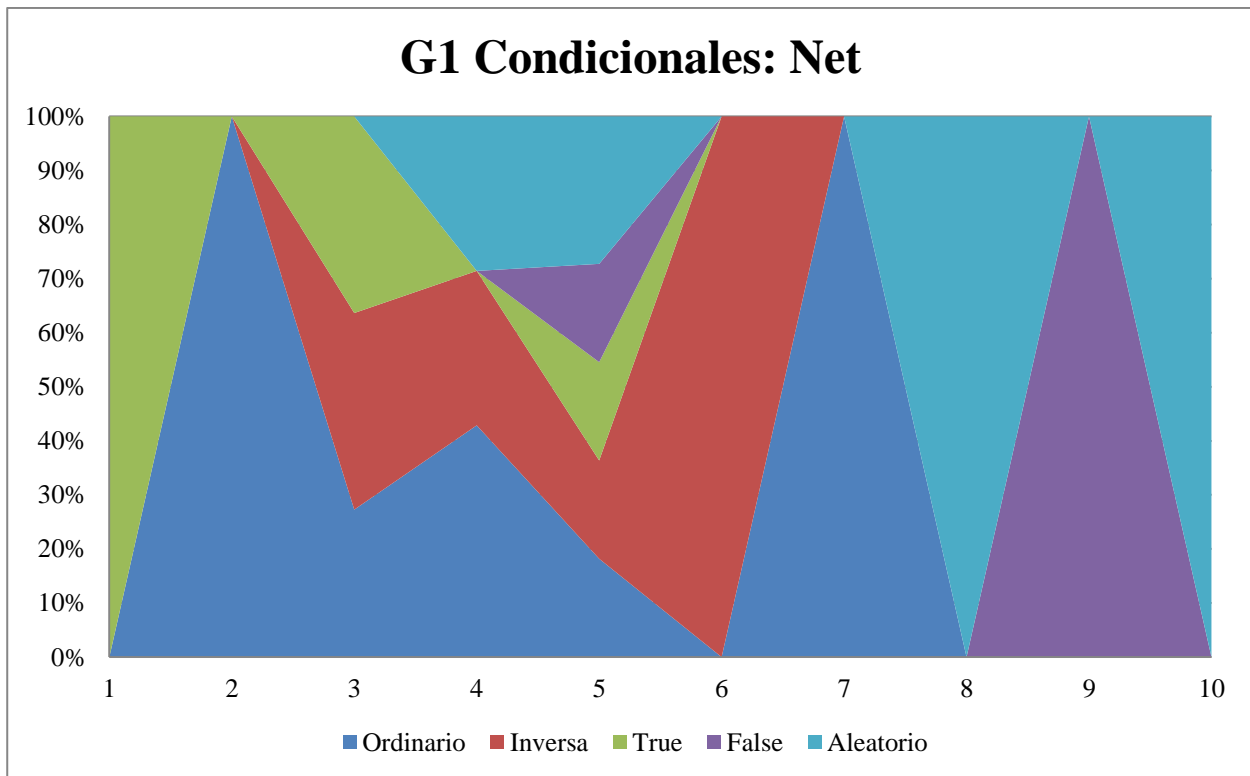


Ilustración 84: G1 Condicionales: Net.

En la gráfica se puede observar las siguientes particularidades:

- Las aplicaciones 2 y 7, en el caso ordinario presentan actividad Net, mientras que en el resto de casos de prueba ninguno.
- La aplicación 6 no ha experimentado variabilidad notoria entre el caso original y los casos de prueba.
- Las aplicaciones 3 y 4, presentan variabilidad en dos tipos de casos de prueba únicamente.

En la Ilustración 85 se muestra el comportamiento de las aplicaciones del Grupo 2:

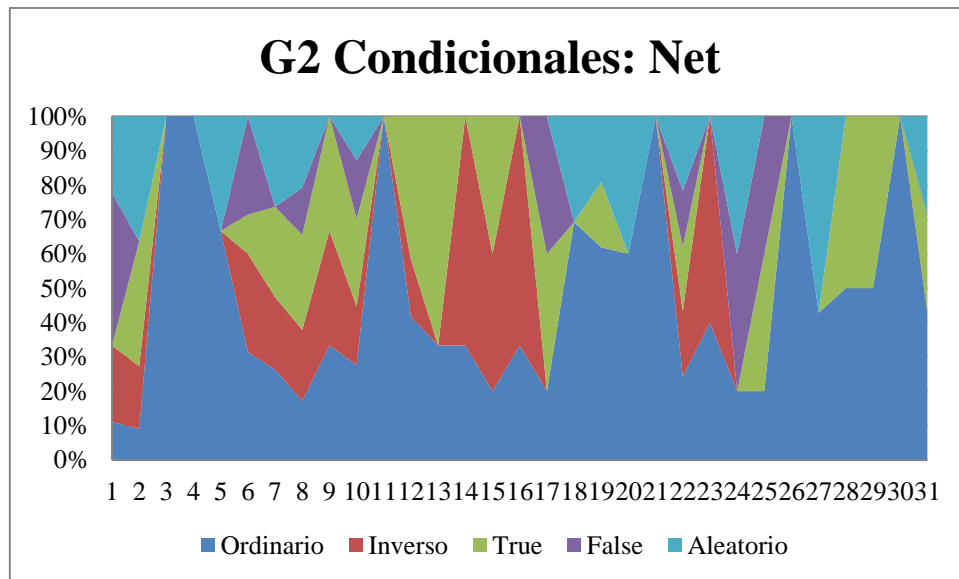


Ilustración 85: G2 Condicionales: Net.

En el 73% de las aplicaciones se puede observar que alguna ejecución de caso de prueba presenta una notoria variación respecto al comportamiento ordinario.

- **CONDICIONALES: LEAKS**

En las aplicaciones del Grupo 2 que presentan este tipo de actividad, se han producido resultados nulos en cada una de las cuatro ejecuciones de caso de prueba, respectivamente.

- **CONDICIONALES: SMS**

En los resultados obtenidos de este grupo de aplicaciones no se han producido operaciones del tipo SMS.

- **CONDICIONALES: CALLS**

En los resultados obtenidos de este grupo de aplicaciones no se han producido operaciones del tipo Calls.

- **CONDICIONALES: DNS**

Como en casos anteriores, se han descartado las aplicaciones cuyos resultados sean nulos tanto en la ejecución ordinaria como en los casos de prueba del tipo condicional.

En esta sección se presentan los niveles de operaciones del tipo *DNS* producidos en casos de prueba del tipo Condicionales.

Las gráficas presentan el porcentaje de *DNS* producidas en las ejecuciones de la aplicación sin modificar (Ordinario) y en las producidas en los casos de prueba basados en invertir las sentencias condicionales presentes en el código (Inversa). También se presentan los porcentajes de *DNS* de los casos de prueba cuyas sentencias condicionales

son ciertas en su totalidad (True), de aquellos casos de prueba en que las sentencias condicionales son todas falsas (False) y casos de prueba que las sentencias condicionales han sido modificadas de manera aleatoria (Aleatorio). El eje de ordenadas presenta el porcentaje de actividad producida en cada tipo de caso de prueba y el eje de abscisas presenta la aplicación en la que se ha producido dicho resultado.

En la Ilustración 86 se muestra el comportamiento relativo a operaciones *DNS* y los casos de prueba del tipo condicional:

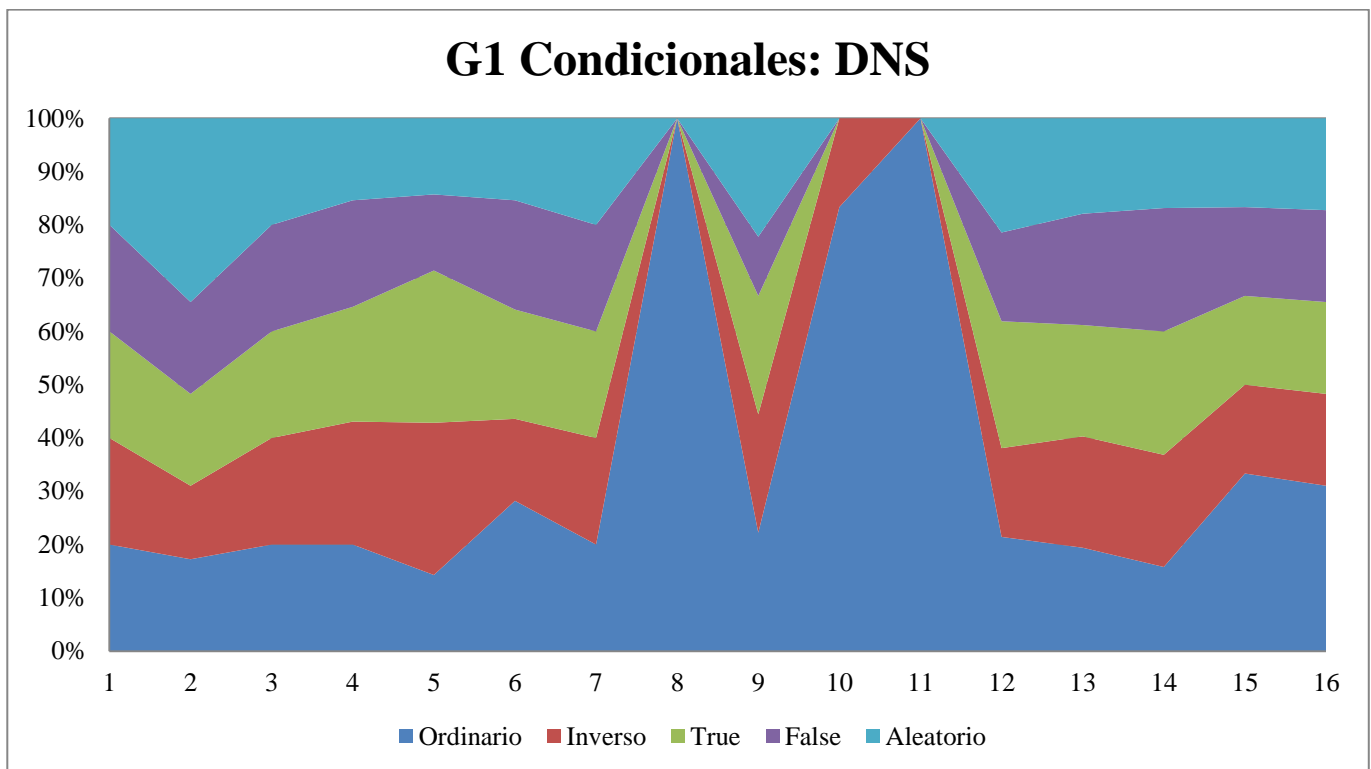


Ilustración 86: G1 Condicionales: DNS.

La gráfica presenta los siguientes rasgos:

- Las aplicaciones 8 y 11 presentan actividad tipo *DNS* en el comportamiento ordinario, pero en los casos de prueba esta actividad es nula.
- La aplicación 10 presenta actividad tipo *DNS* en su caso ordinario, y en el caso de prueba tipo False, aunque el total de operaciones es notoriamente menor respecto al inicial.
- El resto de aplicaciones presenta un número de operaciones *DNS* similar al caso ordinario, con escasa variabilidad en todos los tipos de caso de prueba del tipo condicional.

En la Ilustración 87 se presentan los datos referentes a las aplicaciones del Grupo 2:

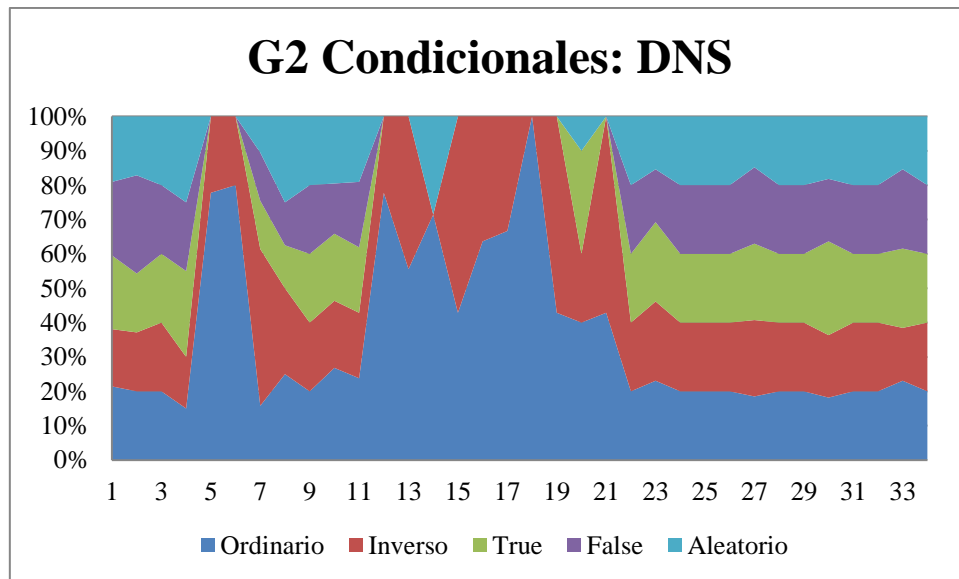


Ilustración 87: G2 Condicionales: DNS.

Se puede contemplar que en un 30% de los resultados obtenidos, en alguna de las ejecuciones de caso de prueba se obtienen resultados notoriamente distintos al comportamiento ordinario de las aplicaciones analizadas.

- **CONDICIONALES: NET CIFRADO**

Se han desechado aquellos casos cuya actividad de tipo *Net cifrado* sea nula tanto en la ejecución de la aplicación original como en cada uno de los casos de prueba.

En esta sección se presentan los niveles de operaciones del tipo *Net cifrado* producidos en casos de prueba del tipo Condicionales.

Las gráficas presentan el porcentaje de actividad *Net cifrado* producida en las ejecuciones de la aplicación sin modificar (Ordinario) y en las producidas en los casos de prueba basados en invertir las sentencias condicionales presentes en el código (Inversa). También se presentan los porcentajes de *Net cifrado* de los casos de prueba cuyas sentencias condicionales son ciertas en su totalidad (True), de aquellos casos de prueba en que las sentencias condicionales son todas falsas (False) y casos de prueba que las sentencias condicionales han sido modificadas de manera aleatoria (Aleatorio). El eje de ordenadas presenta el porcentaje de actividad producida en cada tipo de caso de prueba y el eje de abscisas presenta la aplicación en la que se ha producido dicho resultado.

La Ilustración 88 muestra los resultados obtenidos para este tipo de operación:

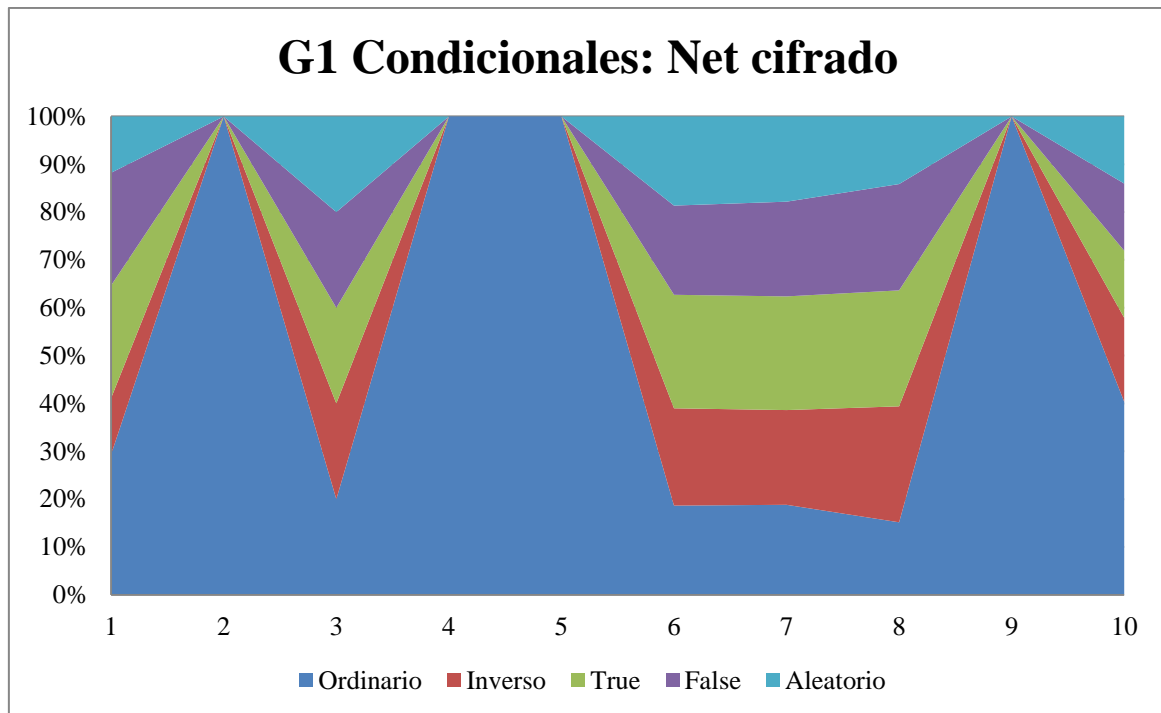


Ilustración 88: G1 Condicionales: Net cifrado.

La gráfica anterior se resume en los siguientes conceptos:

- Las aplicaciones 2, 4, 5 y 9 presentan actividad en su comportamiento ordinario, pero en los distintos casos de prueba esta actividad se convierte en nula.
- El resto de aplicaciones presenta una actividad, en los distintos casos de prueba, cercana o similar a la del caso ordinario.

En la Ilustración 89 se muestran el comportamiento obtenido por las aplicaciones del Grupo 2:

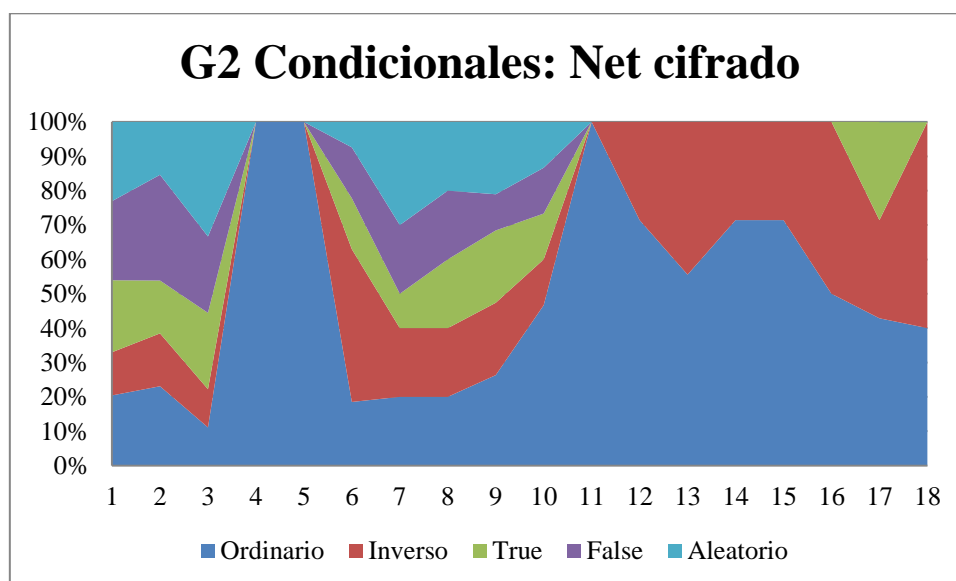


Ilustración 89: G2 Condicionales: Net cifrado.

En un 56% de las aplicaciones analizadas presentan una fuerte variación en alguna de sus ejecuciones de caso de prueba respecto al comportamiento ordinario.

- **CONDICIONALES: NET SIN CIFRAR**

Descartando los resultados de los casos de prueba y comportamiento ordinario sean nulos, se han analizado un total de doce aplicaciones.

En esta sección se presentan los niveles de operaciones del tipo *Net sin cifrar* producidos en casos de prueba del tipo Condicionales.

Las gráficas presentan el porcentaje de actividad *Net sin cifrar* producidas en las ejecuciones de la aplicación sin modificar (Ordinario) y en las producidas en los casos de prueba basados en invertir las sentencias condicionales presentes en el código (Inversa). También se presentan los porcentajes de *Net sin cifrar* de los casos de prueba cuyas sentencias condicionales son ciertas en su totalidad (True), de aquellos casos de prueba en que las sentencias condicionales son todas falsas (False) y casos de prueba que las sentencias condicionales han sido modificadas de manera aleatoria (Aleatorio). El eje de ordenadas presenta el porcentaje de actividad producida en cada tipo de caso de prueba y el eje de abscisas presenta la aplicación en la que se ha producido dicho resultado.

En la Ilustración 90 se resume los resultados obtenidos respecto a este tipo de operación en aplicaciones del Grupo 1:

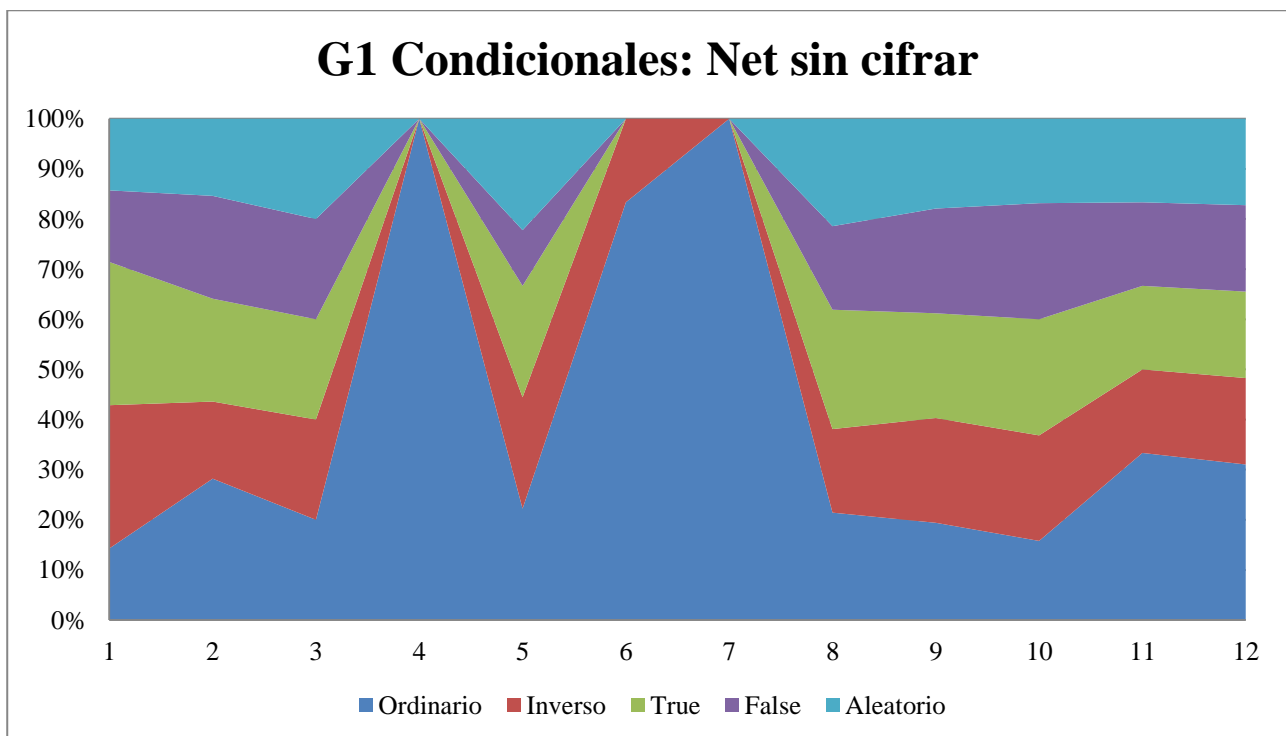


Ilustración 90: G1 Condicionales: Net sin cifrar.

Las principales particularidades representadas en esta gráfica son:

- La aplicación 4 y 7 presentan en sus casos de prueba resultados nulos, y en el caso ordinario sí se encuentra presencia de este tipo de actividad.
- El resto de aplicaciones presenta un porcentaje de actividad en sus casos de prueba similar al del caso ordinario.

En la Ilustración 91 se presenta el mismo tipo de información pero para las aplicaciones del Grupo 2:

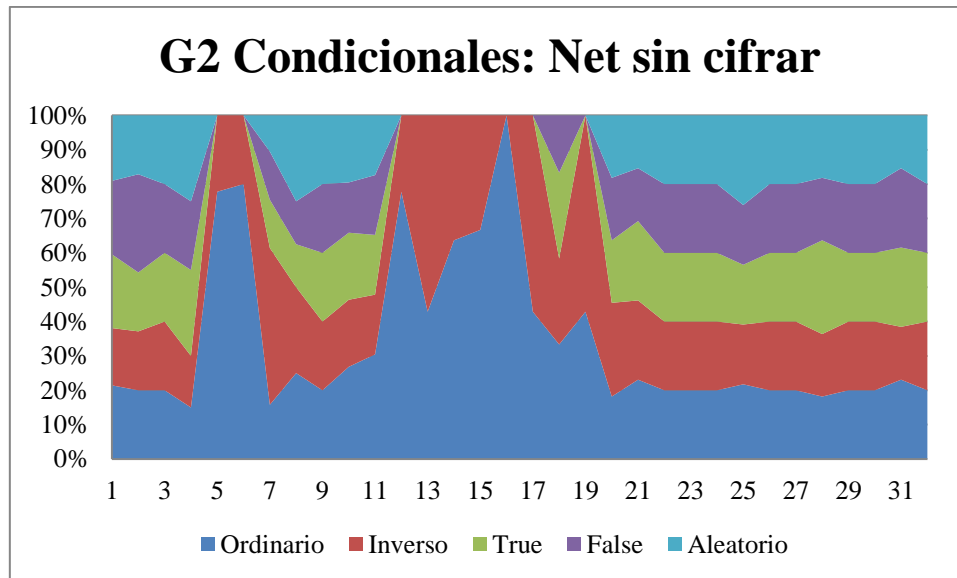


Ilustración 91: G2 Condicionales: Net sin cifrar.

Los resultados se resumen en que en un 32% de las aplicaciones del Grupo 2 que presentan este tipo de actividad, existe una fuerte variabilidad respecto al comportamiento normal de la aplicación en alguna de las ejecuciones de prueba.

• CONDICIONALES: OP CRIPTO

Se han seleccionado las aplicaciones cuyos valores no sean nulos tanto en el comportamiento ordinario como en sus casos de prueba en operaciones del tipo operaciones criptográficas. Se han recopilado un total de cinco aplicaciones de las cien del grupo.

En esta sección se presentan los niveles de operaciones del tipo *Op cripto* producidos en casos de prueba del tipo Condicionales.

Las gráficas presentan el porcentaje de actividad criptográfica producida en las ejecuciones de la aplicación sin modificar (Ordinario) y en las producidas en los casos de prueba basados en invertir las sentencias condicionales presentes en el código (Inversa). También se presentan los porcentajes de *Op cripto* de los casos de prueba cuyas sentencias condicionales son ciertas en su totalidad (True), de aquellos casos de prueba en que las sentencias condicionales son todas falsas (False) y casos de prueba

que las sentencias condicionales han sido modificadas de manera aleatoria (Aleatorio). El eje de ordenadas presenta el porcentaje de actividad producida en cada tipo de caso de prueba y el eje de abscisas presenta la aplicación en la que se ha producido dicho resultado.

En la Ilustración 92 se muestra los resultados obtenidos para el Grupo 1:

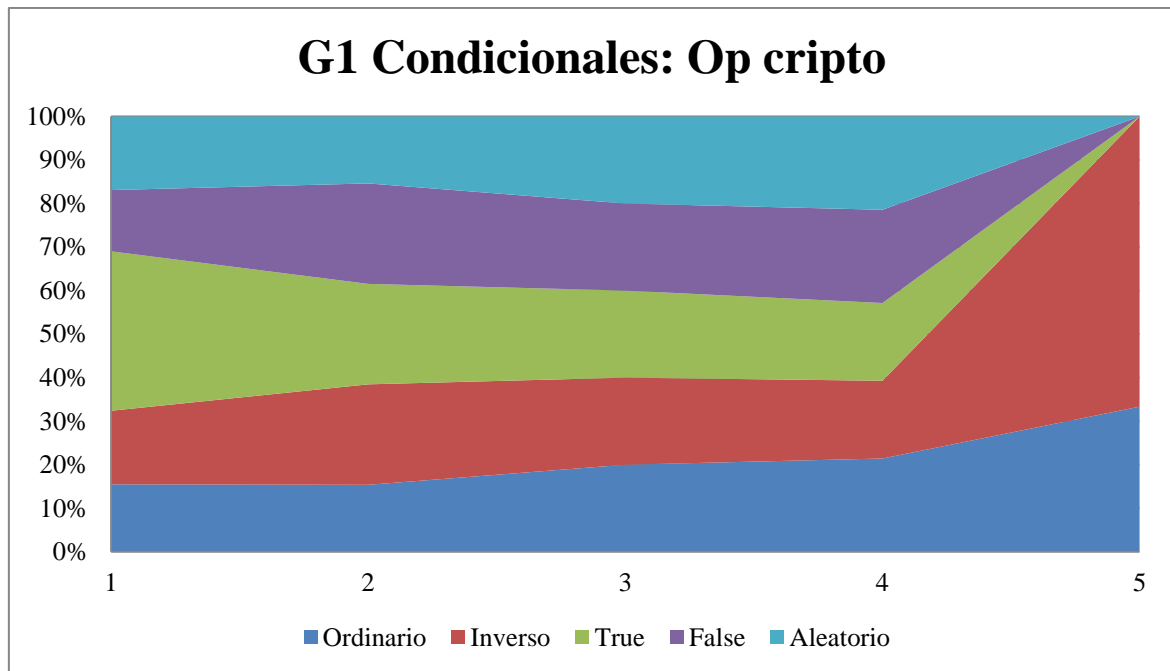


Ilustración 92: G1 Condicionales: Op cripto.

Los principales rasgos recogidos en esta gráfica son:

- La aplicación 5 muestra resultados nulos en todos los casos de prueba excepto en el tipo inverso, que duplica el total de actividad respecto al ordinario.
- El resto de aplicaciones presenta poca variabilidad respecto el caso ordinario, exceptuando el caso de prueba tipo True que presenta un porcentaje más del doble respecto el caso ordinario.

En la Ilustración 93 se muestran los resultados referentes al Grupo 2:

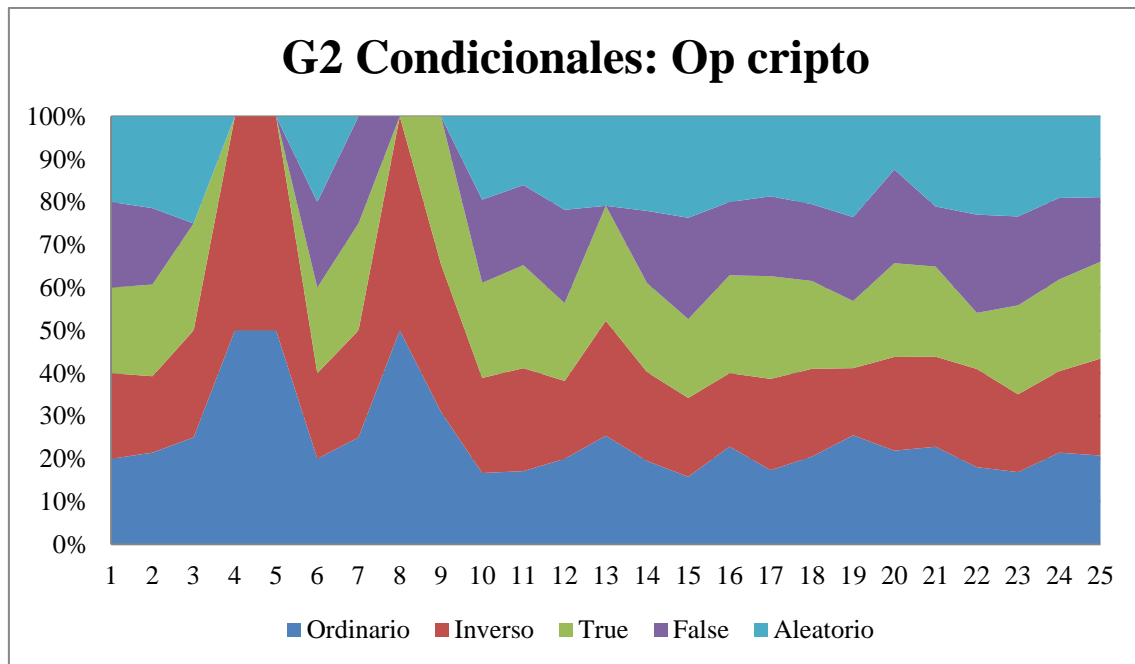


Ilustración 93: G2 Condicionales: Op cripto.

Aproximadamente, en un 28% de las aplicaciones analizadas del Grupo 2 presentan en alguna de las ejecuciones de caso de prueba una considerable variabilidad respecto al comportamiento ordinario detectado.

API CRIPTÓGRAFICA

Este apartado se centra en el comportamiento resultante obtenido en los casos de prueba centrados en analizar las llamadas a APIs criptográficas realizadas por la aplicación.

- **API CRIPTOGRÁFICA: L/E**

En esta sección se presentan los niveles de operaciones del tipo *L/E* producidos en casos de prueba del tipo API criptográfica.

Se han descartado aquellas aplicaciones que no hayan presentado este tipo de operación en su comportamiento.

Las gráficas presentan el porcentaje de actividad *L/E* producidas en la ejecución de la aplicación sin modificar (Ordinario) y en cada una de las ejecuciones de las cadenas de texto analizadas (En esta caso cada aplicación puede presentar de 1 a n ejecuciones, dependiendo del número de cadenas ofuscadas analizadas). El eje de ordenadas presenta el porcentaje de actividad producida en cada ejecución y el eje de abscisas presenta la aplicación en la que se ha producido dicho resultado. Los niveles negativos indican que para dicha aplicación no se ha producido el número de ejecución indicado por la leyenda.

En la Ilustración 94 se muestra el comportamiento relativo a las operaciones del tipo L/E para las aplicaciones del Grupo 1:

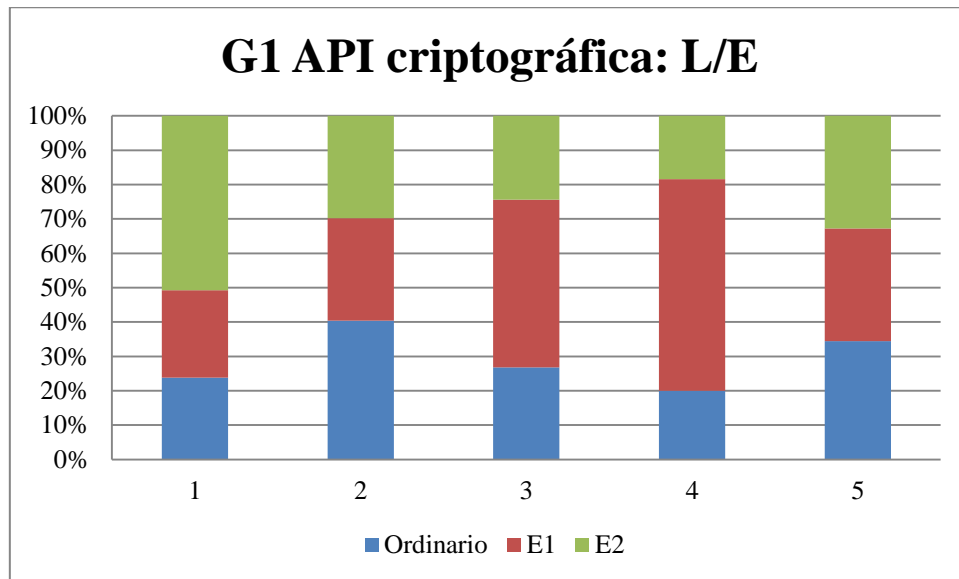


Ilustración 94: G1 API criptográfica: L/E.

Las particularidades producidas durante el análisis son las siguientes:

- En la E2 de la aplicación 1 y en la E1 de las aplicaciones 3 y 4 los resultados doblan el valor de la actividad producida en el comportamiento ordinario.
- El resto de ejecuciones presenta un comportamiento similar al ordinario.

En la Ilustración 95 se presenta el mismo tipo de información pero para las aplicaciones del Grupo 2:

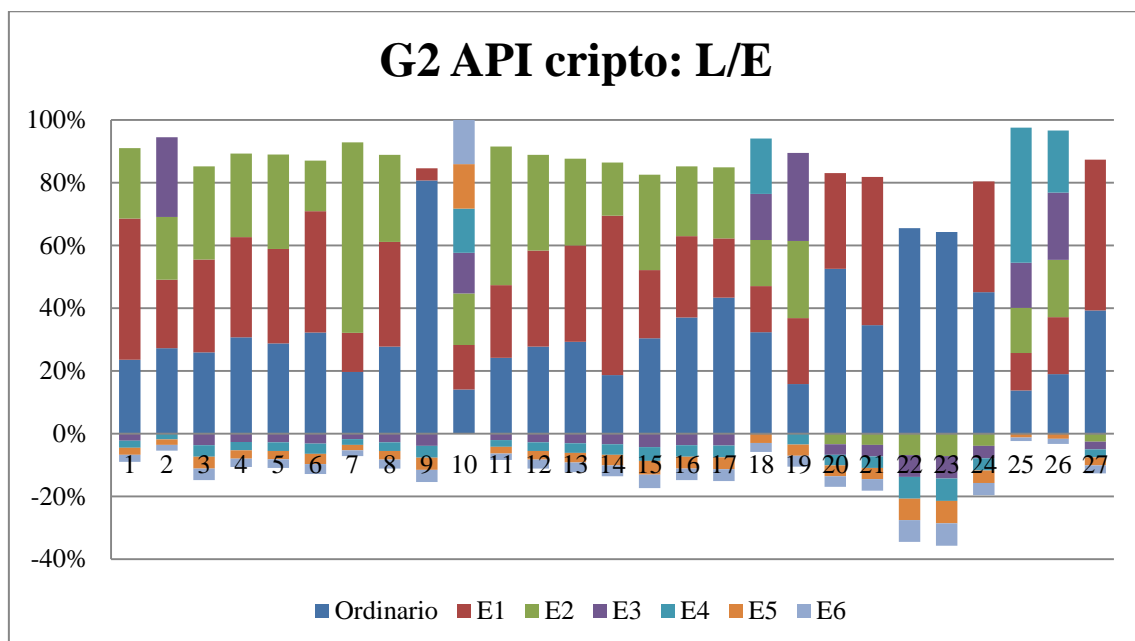


Ilustración 95: G2 API criptográfica: L/E.

En un 20% de las aplicaciones, aproximadamente, presenta en alguna de sus ejecuciones un alto porcentaje de variabilidad respecto al comportamiento ordinario registrado.

- **API CRIPTOGRÁFICA: NET**

En esta sección se presentan los niveles de operaciones del tipo *Net* producidos en casos de prueba del tipo API criptográfica.

Las gráficas presentan el porcentaje de actividad *Net* producidas en la ejecución de la aplicación sin modificar (Ordinario) y en cada una de las ejecuciones de las cadenas de texto analizadas (En esta caso cada aplicación puede presentar de 1 a n ejecuciones, dependiendo del número de cadenas ofuscadas analizadas). El eje de ordenadas presenta el porcentaje de actividad producida en cada ejecución y el eje de abscisas presenta la aplicación en la que se ha producido dicho resultado. Los niveles negativos indican que para dicha aplicación no se ha producido el número de ejecución indicado por la leyenda.

En la Ilustración 96 se presenta el comportamiento de las aplicaciones del Grupo 1:

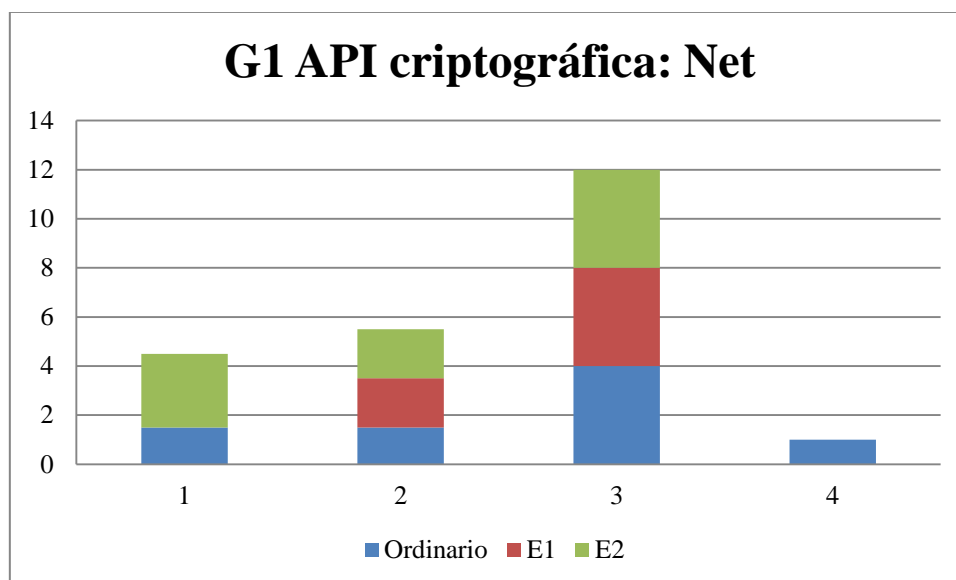


Ilustración 96: G1 API criptográfica: Net.

Los rasgos destacables de estos resultados son:

- En la ejecución E1 de la aplicación 1 y en las ejecuciones E1 y E2 de la aplicación 4 se han obtenido resultados nulos.
- El resto de ejecuciones presentan resultados cercanos al comportamiento ordinario.

El mismo tipo de información es presentado para las aplicaciones del Grupo 2 en la Ilustración 97:

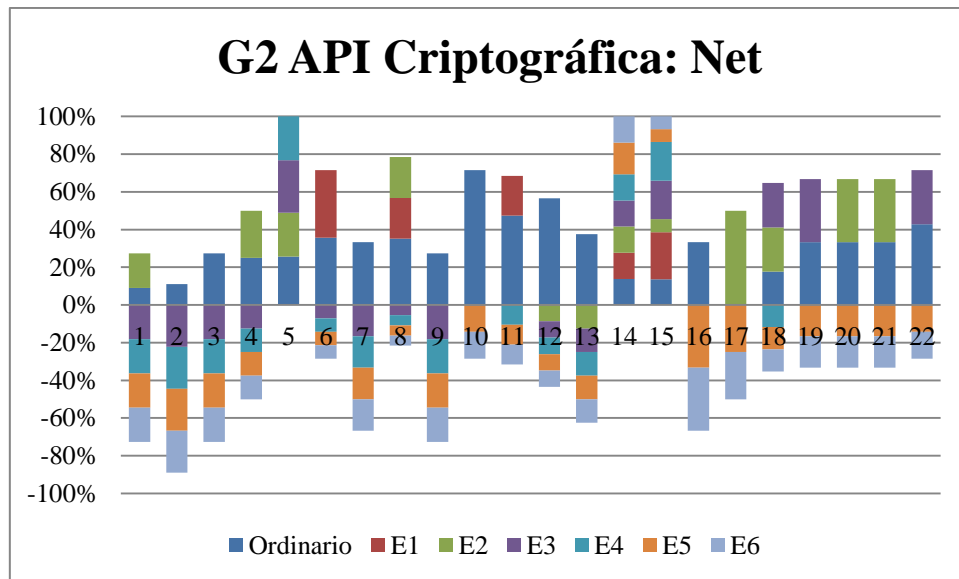


Ilustración 97: G2 API criptográfica: Net.

En un 68% de las aplicaciones analizadas del Grupo 2 se puede observar una fuerte variabilidad, en al menos en una de las ejecuciones, respecto al comportamiento ordinario.

- **API CRIPTOGRÁFICA: LEAKS**

En los resultados obtenidos no se han producido operaciones del tipo *Leaks* en este tipo de caso de prueba.

- **API CRIPTOGRÁFICA: SMS**

En los resultados obtenidos de las aplicaciones analizadas no se han producido operaciones del tipo *SMS*.

- **API CRIPTOGRÁFICA: CALLS**

En los resultados obtenidos de las aplicaciones analizadas no se han producido operaciones del tipo *Calls*.

- **API CRIPTOGRÁFICA: DNS**

En esta sección se presentan los niveles de operaciones del tipo *DNS* producidos en casos de prueba del tipo API criptográfica.

Las gráficas presentan el porcentaje de actividad *DNS* producidas en la ejecución de la aplicación sin modificar (Ordinario) y en cada una de las ejecuciones de las cadenas de texto analizadas (En esta caso cada aplicación puede presentar de 1 a n ejecuciones, dependiendo del número de cadenas ofuscadas analizadas). El eje de ordenadas presenta el porcentaje de actividad producida en cada ejecución y el eje de abscisas presenta la aplicación en la que se ha producido dicho resultado. Los niveles negativos indican que

para dicha aplicación no se ha producido el número de ejecución indicado por la leyenda.

En la Ilustración 98 se muestra los resultados obtenidos tras los casos de prueba del tipo API criptográfica centrados en operaciones del tipo *DNS* de las aplicaciones del Grupo1:

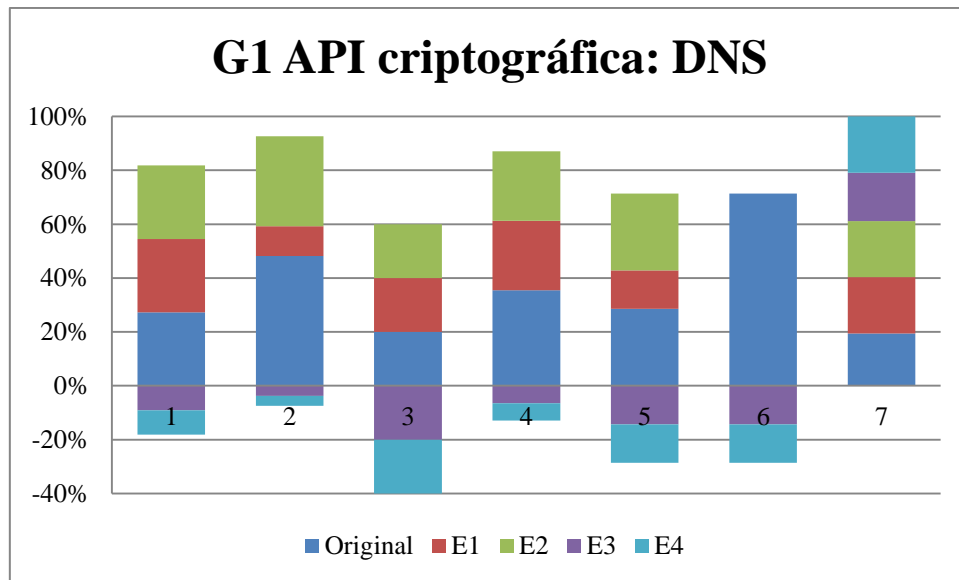


Ilustración 98: G1 API criptográfica: DNS.

Los valores negativos presentan ausencia de ejecución para dicha aplicación.

Los principales rasgos a destacar son:

- La aplicación 6 presenta resultados nulos en todas sus ejecuciones.
- El resto de ejecuciones en las distintas aplicaciones presentan valores similares al comportamiento ordinario.

Los resultados obtenidos del Grupo 2 son presentados en la Ilustración 99:

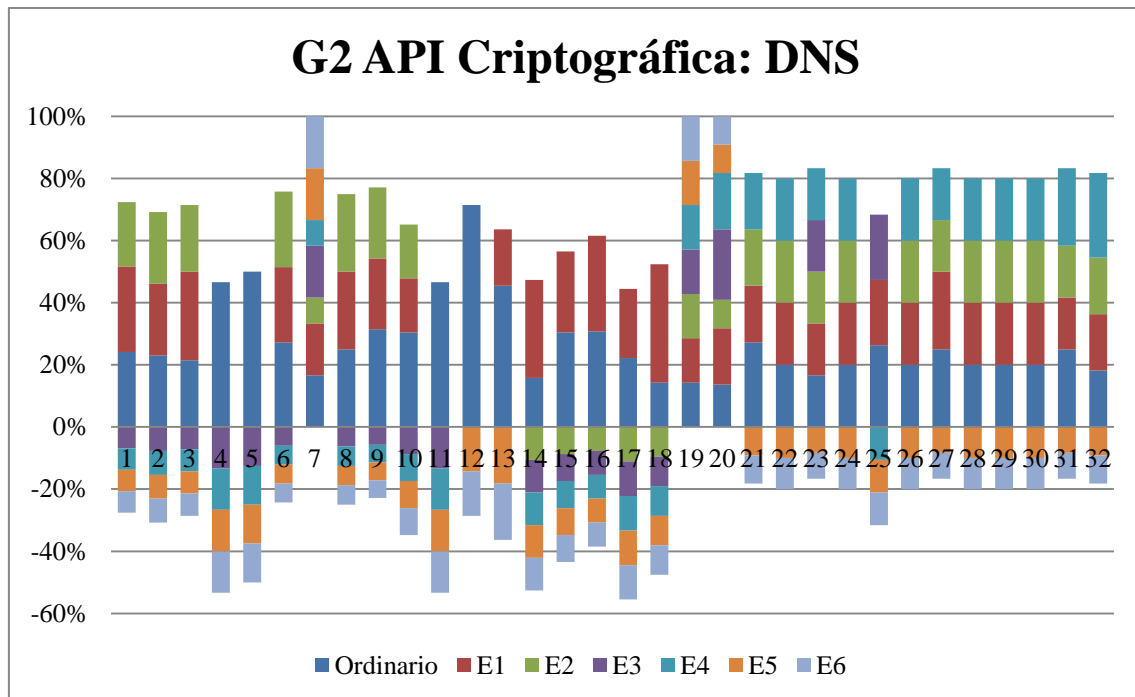


Ilustración 99: G2 API criptográfica: DNS.

- **API CRIPTOGRÁFICA: NET CIFRADO**

En la Ilustración 100 se muestra los resultados del Grupo 1 obtenidos tras los casos de prueba del tipo API criptográfica centrados en operaciones del tipo *Net cifrado*:

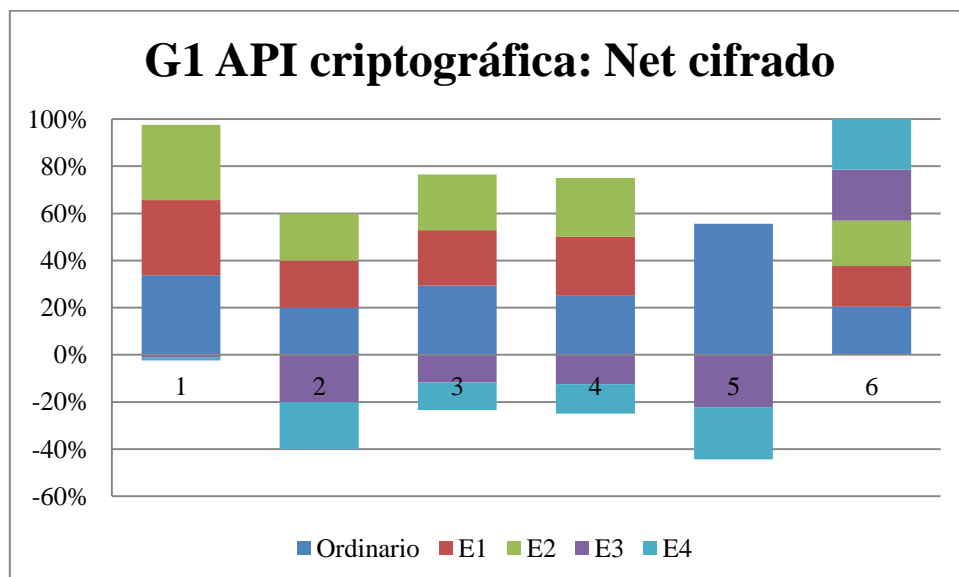


Ilustración 100: G1 API criptográfica: Net cifrado.

Los valores negativos presentan ausencia de ejecución para dicha aplicación.

Los principales datos a destacar son:

- La aplicación 5 presenta resultados nulos en todas sus ejecuciones.
- El resto de ejecuciones en las distintas aplicaciones presentan valores similares al comportamiento ordinario.

En la Ilustración 101 se presentan los resultados del Grupo 2:

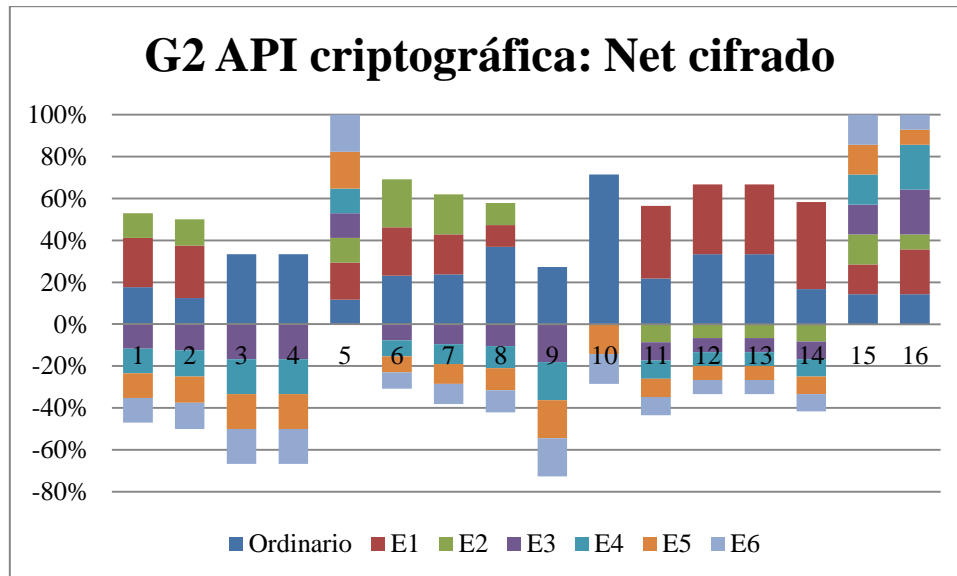


Ilustración 101: G2 API criptográfica: Net cifrado.

Aproximadamente, en un 32% de las aplicaciones analizadas del Grupo 2 presentan una fuerte variabilidad en alguna de sus ejecuciones de caso de prueba respecto al comportamiento ordinario.

• API CRIPTOGRÁFICA: NET SIN CIFRAR

En esta sección se presentan los niveles de operaciones del tipo *Net sin cifrar* producidos en casos de prueba del tipo API criptográfica.

Las gráficas presentan el porcentaje de actividad *Net sin cifrar* producidas en la ejecución de la aplicación sin modificar (Ordinario) y en cada una de las ejecuciones de las cadenas de texto analizadas (En esta caso cada aplicación puede presentar de 1 a n ejecuciones, dependiendo del número de cadenas ofuscadas analizadas). El eje de ordenadas presenta el porcentaje de actividad producida en cada ejecución y el eje de abscisas presenta la aplicación en la que se ha producido dicho resultado. Los niveles negativos indican que para dicha aplicación no se ha producido el número de ejecución indicado por la leyenda.

En la Ilustración 102 se muestra los resultados obtenidos tras los casos de prueba del tipo API criptográfica centrados en operaciones del tipo Net sin cifrar:

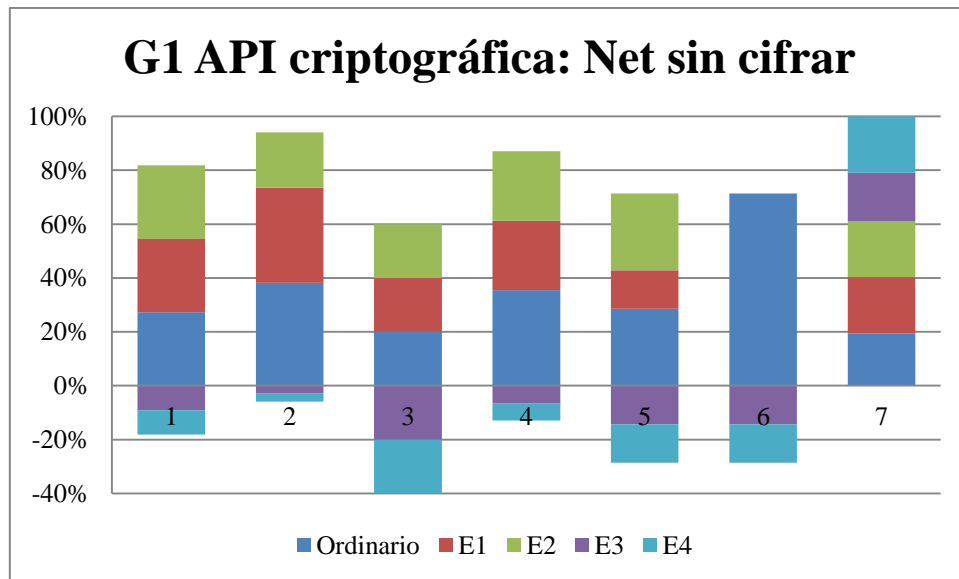


Ilustración 102: G1 API criptográfica: Net sin cifrar.

Los valores negativos presentan ausencia de ejecución para dicha aplicación.

Las principales particularidades de los resultados obtenidos son:

- La aplicación 5 presenta resultados nulos en todas sus ejecuciones.
- El resto de ejecuciones en las distintas aplicaciones presentan valores similares al comportamiento ordinario.

En la Ilustración 103 se presentan los resultados obtenidos de las aplicaciones del Grupo 2:

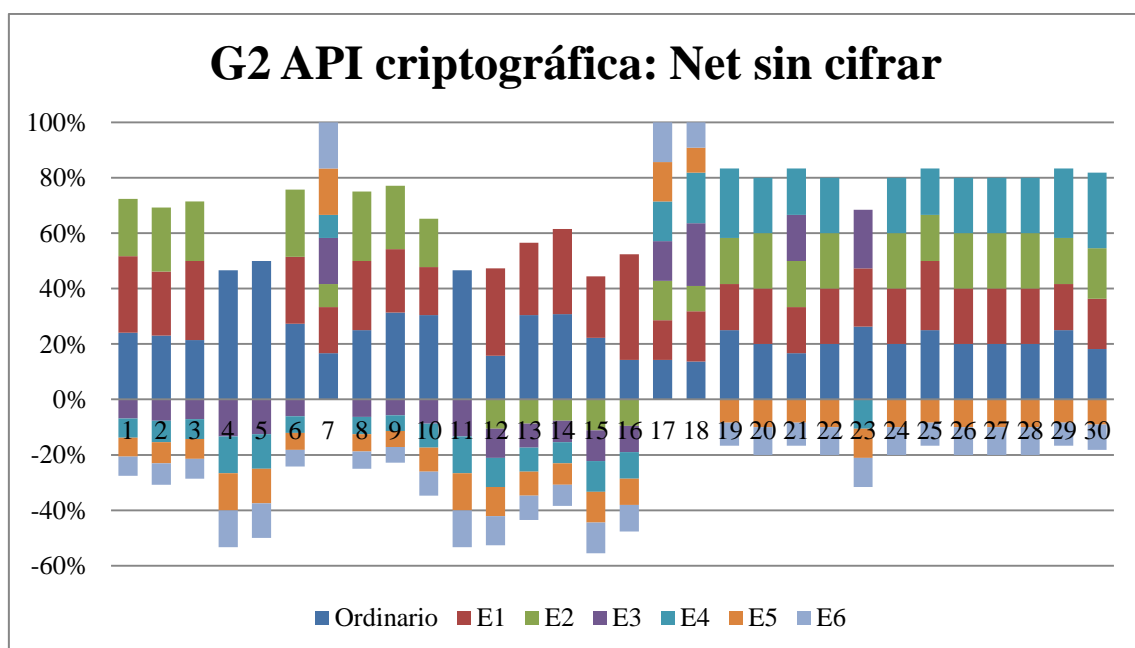


Ilustración 103: G2 API criptográfica: Net sin cifrar.

En un 46% de las aplicaciones analizadas del Grupo 2 se han obtenido resultados que presentan una notoria variabilidad en alguno de sus casos de prueba de este tipo frente al comportamiento ordinario.

• API CRIPTOGRÁFICA: OP CRIPTO

En esta sección se presentan los niveles de operaciones del tipo *Op crypto* producidos en casos de prueba del tipo API criptográfica.

Las gráficas presentan el porcentaje de actividad *Op crypto* producidas en la ejecución de la aplicación sin modificar (Ordinario) y en cada una de las ejecuciones de las cadenas de texto analizadas (En esta caso cada aplicación puede presentar de 1 a n ejecuciones, dependiendo del número de cadenas ofuscadas analizadas). El eje de ordenadas presenta el porcentaje de actividad producida en cada ejecución y el eje de abscisas presenta la aplicación en la que se ha producido dicho resultado. Los niveles negativos indican que para dicha aplicación no se ha producido el número de ejecución indicado por la leyenda.

En la Ilustración 104 se presentan los resultados obtenidos relativos al Grupo 1 tras los casos de prueba del tipo API criptográfica centrados en las operaciones criptográficas producidas:

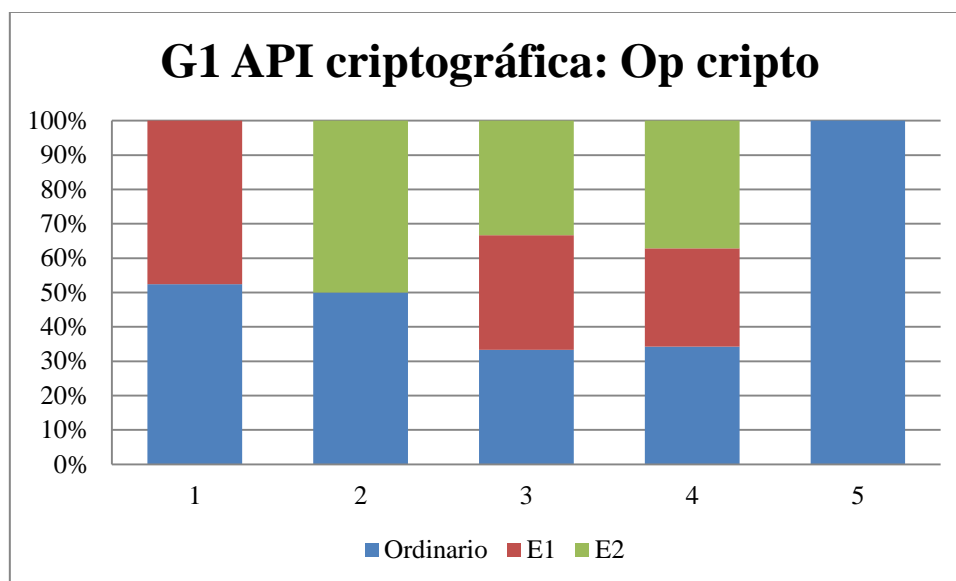


Ilustración 104: G1 API criptográfica: Op crypto.

Los principales rasgos a destacar son:

- En las ejecuciones E1 y E2 de la aplicación 5 no se producen actividad de este tipo. En la ejecución E1 de la aplicación se da el mismo caso.
- El resto de aplicaciones presente niveles de actividad criptográfica similares al del caso ordinario.

En la Ilustración 105 se presentan los resultados correspondientes al Grupo 2:

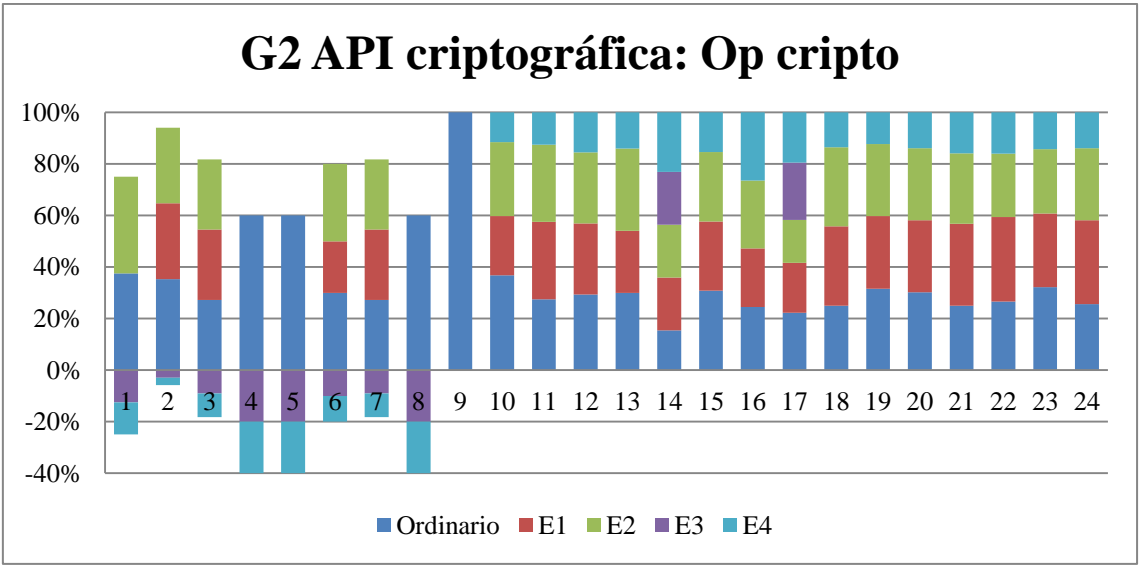


Ilustración 105: G2 API criptográfica: Op cripto.

En un 75% de las aplicaciones del Grupo 2 que presentan actividad criptográfica, se han obtenido resultados completamente distintos en alguna de las ejecuciones de caso de prueba respecto al comportamiento ordinario de la aplicación