



Escuela Técnica Superior de Ingeniería Informática

Departamento de Arquitectura y Tecnología de Computadores,
Ciencia de la Computación e Inteligencia Artificial

Tesis Doctoral:

**Una arquitectura distribuida para la detección,
comunicación y mitigación de la denegación de
servicio**

Tesis presentada por Luis Campo Giralte para obtener el grado de
Doctor en Informática
por la Universidad Rey Juan Carlos

Dirigida por:
Enrique Cabello Pardos
Coodirigida por:
Cristina Conde Vilda

*There are only two things to remember.
Number one, don't stop and number two, keep going.
Frank Zappa.*

Agradecimientos

Este trabajo no habría sido posible sin el apoyo y el estímulo de las siguientes personas:

- Federico Castanedo, por animarme desde el fin de nuestra carrera universitaria a adentrarme en el mundo de los artículos.
- Ricardo Jimenez Peris y Marta Patiño, por brindarme la oportunidad de trabajar con ellos en la Universidad Politécnica de Madrid y aprender sistemas distribuidos con ellos.
- Mario López, por haberme facilitado trazas de tráfico de valor incalculable para esta tesis, sin las cuales ésta no habría sido posible.
- Eduardo Bermúdez, por su ayuda en la edición de diversos artículos.
- Cristina Conde e Isaac Martín, por sus continuas revisiones de los artículos y de esta tesis.
- A Enrique Cabello, por darme la oportunidad de trabajar con él en esta tesis a lo largo de estos años.
- A Victor Martínez Maestro, por ofrecerme una guía espiritual y humana de la vida.
- A Pilar Segarra Catasús, por ayudarme en las continuas revisiones de esta tesis y por servirme de apoyo vital en mi vida.
- A mis padres por servirme de continuo referente para lidiar con los problemas de la vida, por que es a ellos a quien dedico esta tesis.

Resumen

Como bien se sabe, en la actualidad los ciberataques tienen lugar en cualquier lugar del mundo y afectan a todo tipo de infraestructuras. Todos los meses se reciben noticias de ciberataques que se llevan a cabo contra grandes empresas, gobiernos, universidades, comercios de Internet, entre otros, y ninguna de las soluciones comerciales disponibles parece capaz de ofrecer una solución al gran problema de la denegación de servicio.

Es por ello que, con el objeto de ofrecer una adecuada protección frente a dichas intrusiones, resulta necesario aportar soluciones configurables, integrables que, al mismo tiempo, sean rápidas y eficaces. En este sentido, la tesis que se expone a continuación presenta y desarrolla una solución de arquitectura distribuida para la detección, comunicación y mitigación de la denegación del servicio, basada en aplicaciones software híbridas desarrolladas en un núcleo en lenguaje C, así como en la flexibilidad que aporta Python, empleando para ello interfaces externas en dicho lenguaje.

En esta tesis se exploran soluciones a la denegación de servicio a nivel de aplicación. Para ello, se han estudiado e investigado técnicas basadas en redes neuronales, en análisis estadísticos a nivel IP, en correlación de flags TCP, en el estudio de comportamientos del usuario sobre los recursos del servidor, soluciones de control de flujo y en soluciones distribuidas, entre otras. El resultado de las técnicas estudiadas ha dado lugar a una solución distribuida, escalable y de bajo coste, dependiente del tipo de infraestructura que se quiere proteger.

La propuesta considera una arquitectura distribuida basada en tres pilares fundamentales, detección, comunicación y mitigación de la denegación de servicio. Dichos elementos, cooperan entre sí de manera que, tanto en datos reales y simulados, conseguimos detectar, mitigar y comunicar este tipo de ciberataques. Para la detección proponemos el empleo de un algoritmo que aporta elevadas tasas de detección en el protocolo HTTP. Por otra parte, mediante el uso de canales encubiertos conseguimos comunicar los diferentes elementos de la arquitectura propuesta de forma prácticamente invisible. Finalmente, conseguimos mitigar dichas intrusiones de manera que los usuarios habituales del servicio no experimenten ninguna alteración en sus comunicaciones.

Todo lo que aquí se expone contiene código fuente con licencia GPL publicado en Internet y puede ser descargado y utilizado por otros investigadores si así fuese necesario.

Abstract

As is well known, at present cyber attacks occur in any part of the world and affect all types of infrastructure. Every month news of cyber attacks on large corporations, governments, universities, web sites, and none of the commercial solutions available on the market seems to provide a solution to the great problem of the denial of service.

For this reason, in order to provide adequate protection against such intrusions, it is necessary to provide solutions which are configurable, easy to integrate and fast and effective at the same time. In this sense, the following thesis presents and develops a distributed architecture solution for detection, communication and mitigation of denial of service based on hybrid software applications, which are develop in C language and all the external interfaces develop in Python language in order to bring more flexibility to the proposal.

In this thesis we explore solutions to the denial of service at application level. We have studied the following techniques such as based on neural networks, based on IP statistical values, on TCP flags correlation, on web server user behaviour, based on flow control, distributed solutions and so on. Taking into account these techniques we come to a distributed, scalable and low cost solution depending on the infrastructure to protect.

The proposal considers a distributed architecture based on three fundamental pillars: detection, communication and mitigation of denial of service. These elements, cooperate with each other so that, by actual test data and simulated we detect, mitigate and communicate this type of cyber attacks. For the detection we propose the use of an algorithm that provides high detection rates in the HTTP protocol. Moreover, by using network covert channels we communicate the different elements of the architecture proposal making them practically invisible. Finally, we mitigate the attacks so that regular users of the service do not experience any alteration in their communications.

Everything presented here contains GPL source code published on the Internet in which you can download and use to other researchers purposes if needed.

Índice general

1. Introducción	9
1.1. Introducción y motivación	9
1.2. Objetivo de la tesis	9
1.3. Planteamiento del problema y solución propuesta	10
1.4. Estructura de la memoria	11
2. Estado del arte	12
2.1. Introducción	12
2.2. Anatomía de un ataque	15
2.3. Modelos de prevención	18
2.3.1. Detección y neutralización de Host secundarios	18
2.3.2. Detección y prevención de víctimas	19
2.3.3. Detección y prevención de ataques potenciales	20
2.3.4. Mitigación y detención de ataques	22
2.3.5. Desvío de ataques	25
2.3.6. Análisis forense	26
2.4. Modelos de detección	27
2.5. Modelos de comunicación	30
2.6. Modelos de mitigación	33
2.7. Necesidades y carencias actuales	37
3. Arquitectura propuesta	39
3.1. Esquema de la solución	40
3.2. Integración en la red real	45
3.3. Conclusiones	47
4. Detección de ataques DDoS	48
4.1. Planteamiento del problema	48
4.2. Solución	49
4.2.1. Análisis estadístico	51
4.2.2. Análisis de la cache de grafo HTTP (CGH)	52
4.2.3. Análisis de la cache de caminos HTTP (CCH)	55
4.3. Experimentos y resultados	60

<i>Índice general</i>	8
4.3.1. Modelo de tráfico del servidor web cacheado	60
4.3.2. Comportamiento del modelo frente ataques	63
4.4. Conclusiones	65
5. Comunicaciones seguras	66
5.1. Planteamiento del problema	66
5.2. Solución	67
5.2.1. Descripción del protocolo	67
5.2.2. Port walking	69
5.2.3. Payload noise	71
5.2.4. IP randomness	74
5.2.5. Signature poisoning	75
5.2.6. Implementación	76
5.3. Experimentos y resultados	78
5.4. Conclusiones	82
6. Mitigación del DDoS	83
6.1. Planteamiento del problema	83
6.2. Solución	84
6.2.1. Funcionalidad software	84
6.2.2. Características software	85
6.3. Implementación	86
6.4. Experimentos y resultados	91
6.4.1. Colas lineales	94
6.4.2. Colas exponenciales	96
6.5. Conclusiones	99
7. Conclusiones	100
7.1. Aportaciones de la tesis	103
7.2. Publicaciones	103
7.3. Trabajos futuros	104

Capítulo 1

Introducción

1.1. Introducción y motivación

Desde hace años los ataques a las redes de Internet, especialmente aquellos conocidos como DDoS (*Distributed Denial of Service*), han sido objeto de mi interés, tanto académico como profesional. Si bien este tipo de agresiones vienen produciéndose desde el origen de Internet, al menos hasta la fecha no parecen existir soluciones eficientes que garanticen la detección, mitigación y comunicación de dichos ataques.

Conviene también destacar el elevado coste económico que este tipo de agresiones implican, así como la variedad de usuarios a los que puede ir dirigidos como, por ejemplo, comercios electrónicos que verán afectados sus servicios, proveedores de contenidos incapaces de entregar sus contenidos a los clientes o centrales nucleares que pierden la conexión con el exterior, entre otros muchos. Los aspectos mencionados, junto con otros, convierten a Internet en un ámbito inseguro en el que absolutamente ningún usuario se encuentra a salvo de este tipo de ataques y de sufrir sus consecuencias.

En este sentido, durante estos años he procurado seguir, y así lo continuo haciendo, todas aquellas noticias relacionadas con este tipo de agresiones, especialmente aquellas relativas al modo en que se producían los ataques, a los diversos tipos de protocolos implicados, a los diferentes procedimientos para la programación de una ciber herramienta de ataque, a los tipos de infraestructuras afectadas o a las diversas estrategias llevadas a cabo para materializarlos, entre otras. Todas ellas inquietudes que han despertado mi interés para desarrollar esta tesis.

1.2. Objetivo de la tesis

El principal objetivo del documento que se presenta consiste en la realización de un sistema totalmente distribuido que permita la detección, comunicación y mitigación de la denegación de servicio distribuida. La detección se encargará de

determinar cuando se ha producido un ataque y cuando no, la comunicación se responsabilizará de la notificación de dicha intrusión (a otros elementos) y, finalmente, la mitigación adoptará medidas activas en la red o en los nodos subyacentes con el objeto de detener, en la mayor medida posible, el impacto del ataque.

No obstante, para la realización de esta tesis también se ha considerado prioritario aportar una solución de bajo coste, tanto en términos económicos como de computación. Por otra parte, dado que la mayoría de soluciones existentes para este tipo de ataques emplean código cerrado y, por ello, poco verificable por la comunidad científica, se ha procurado que el código fuente de las diferentes partes que integran la solución propuesta sea un código abierto, de libre acceso, verificable y extensible a otras aplicaciones.

1.3. Planteamiento del problema y solución propuesta

El problema de la denegación del servicio continua siendo un tema de actualidad, ampliamente abordado por diversos autores y productos comerciales cuyo principal objetivo reside en hacer frente a los continuos ataques, a nivel mundial, que se vienen produciendo en cualquier tipo de organización, ya sean compañías estatales, privadas u otras.

Dificultades esperadas del problema

La presente tesis propone una solución de bajo coste, implementable en los nodos de la red que abarca las fases de detección de ataques, de comunicación segura de los mismos y de mitigación de sus efectos.

Su ámbito de actuación queda enmarcado en una campo extremadamente dinámico, en constante evolución y de considerable opacidad dados los diversos intereses económicos y de seguridad. Es por ello que, ante la dificultad para la obtención de información real de los ataques producidos, esta tesis aporta una solución basada tanto en el empleo de datos obtenidos mediante simulaciones de laboratorio, como en la utilización de información real, siempre que haya sido posible disponer de la misma.

Otro de los problemas con que nos hemos encontrado ha consistido en la complejidad de gestionar un ataque de forma realista y con recursos limitados, especialmente si se considera que en un ataque real de denegación de servicio pueden estar implicados más de 10.000 equipos conectados a Internet. Finalmente, también es conveniente resaltar la dificultad para disponer de logs o trazas de tráfico real que nos permitieran realizar la simulación de ataques a una infraestructura con el objeto de estudiar su impacto en los usuarios legítimos.

Solución propuesta

En esta tesis aportamos una solución híbrida, basada en la distribución de los elementos que forman parte de la arquitectura (detección, comunicación y mitigación), capaz de soportar el crecimiento y el despliegue en diferentes arquitecturas de red, ya sea en tanto en términos de coste económico como en términos generales. La solución propuesta reúne las siguientes características:

- Flexibilidad, con el objeto de aportar una solución capaz de soportar cambios en la configuración y parametrización del software.
- Robustez, con la finalidad de permitir que el modelo implementado disponga de capacidad suficiente para soportar elevadas cargas de trabajo de red.
- Integración, de manera que el software pueda colaborar con otros sistemas de seguridad, Firewalls, NIDS, sistemas de autenticación, etc...
- Optimización de las estructuras y de la complejidad, con la finalidad de generar un código con calidad suficiente que permita gestionar millones de paquetes IP y de flujos de red.

1.4. Estructura de la memoria

La presente memoria está organizada en varios capítulos. En el capítulo 2 realizamos un repaso del estado de arte de las soluciones que proponen los autores mas relevantes. Seguidamente en el capítulo 3 exponemos la arquitectura del sistema completo. A continuación, en el capítulo 4 analizamos el sistema de detección de ataques. En el capítulo 5 mostramos un sistema de comunicación invisible. Posteriormente, en el capítulo 6 estudiamos el sistema de mitigación de ataques y, finalmente, en el capítulo 7, exponemos las conclusiones globales acerca del sistema final, las aportaciones de esta tesis y proponemos algunos caminos futuros para la exploración de este tipo de ataques.

Capítulo 2

Estado del arte

2.1. Introducción

A lo largo de la historia de Internet las denegaciones de servicio han sido un problema contra lo que no se ha podido luchar de una manera eficiente.

De hecho, uno de los principales problemas radica en la arquitectura de Internet basada en sistemas extremo a extremo, en los que un Host puede recibir paquetes IP sin haberlos solicitado, que además pueden ser empleados con fines ilícitos y en los que el Host no dispone de mecanismos para detener los paquetes que no desea recibir.

Inicialmente, en los primeros ataques distribuidos existían varios atacantes contra una maquina los cuales enviaban paquetes IP con direcciones origen falsas (*Spoofing*). Este tipo de ataques se producían principalmente en modo datagrama, de manera que las direcciones IP al ir falseadas en origen, únicamente permitían la utilización de protocolos de nivel superior que funcionaban en modo datagrama, como pueden ser ICMP, UDP o paquetes SYN TCP.

Otro tipo de intrusión que modificó el modo de realización de los ataques de denegación de servicio fue el ataque directo DDoS, concretamente mediante la herramienta Trin00 [1], también conocida como TFN. Originalmente este tipo de ataque se localizó en forma binaria en sistemas Solaris 2.X y se compone de un conjunto de programas clientes y demonios que implementan una herramienta de denegación de servicio distribuida, capaz de producir ataques por generación masiva de paquetes ICMP, TCP SYN o UDP, así como ataques del tipo Smurfing [2].

TFN2K [3] es una evolución del anteriormente comentado TFN. En éste, se denomina maestro al sistema informático en el que corre el cliente, y agente al sistema informático donde se ejecuta el demonio. El TFN2K permite a los maestros explotar los recursos de un determinado número de agentes con el fin de coordinar el ataque a una o más víctimas. El maestro configura a los agentes para atacar a una determinada lista de víctimas por inundación de paquetes IP. En este tipo de ataque se establecen dos flujos de datos. El primero de

ellos consiste en comandos de control con los que se controlan a los huéspedes que realizan el ataque. Dichos huéspedes son capaces de enviar tanto paquetes falseados como conexiones TCP completas, ya que la dirección del atacante en ningún momento es conocida, salvo por el uso de ciertos campos en los paquetes ICMP, como realiza Trino y sus sucesores. El segundo flujo de datos corresponde al ataque propiamente dicho.

Stacheldraht [4] fue detectada entre finales de septiembre y comienzos de octubre de 1999, tanto en sistemas europeos como norteamericanos. Stacheldraht, término de origen alemán que podría traducirse por alambre de espino o alambrada de espinos, combina características de Trino y TFN e incorpora mecanismos de cifrado en la comunicación entre el cliente y el servidor, así como mecanismos de actualización automática de los agentes.

Otra de las más recientes herramientas de DoS ha sido Slowloris [5, 6], que fue utilizada en un ataque DDoS contra instalaciones en Iran [7]. A continuación se muestra como estaría formada la petición HTTP [8] principal de dicha herramienta.

```
GET / HTTP/1.1
Host: 192.168.1.254
User-Agent: Mozilla/4.0 (compatible; MSIE 7.0; Windows NT 5.1)
Content-Length: 42
X-a: b
```

Como puede observarse en la petición HTTP, al no pertenecer el parámetro *X-a* al estándar HTTP, cualquier sistema que inspeccionara dicha petición podría detectar fácilmente el uso de Slowloris e intentar mitigar dicho ataque.

Sin embargo, no todas las herramientas dejan rastro como ocurre con Slowloris. Por ejemplo, la herramienta conocida como Hulk [9], implementada en Python, utiliza el campo *User-Agent* de HTTP de manera aleatoria junto con otros campos más de forma que, como puede observarse en el fragmento de código 2.1 de Hulk, al emplear valores conocidos de campos HTTP para que ésta no pueda ser detectada por las técnicas de DPI (*Deep Packet Inspection*).

```

def httpcall(url):
    useragent_list()
    referer_list()
    code=0
    if url.count("?")>0:
        param_joiner="&"
    else:
        param_joiner="?"
    request = urllib2.Request(url + param_joiner + buildblock(random.
        randint(3,10)) + '=' + buildblock(random.randint(3,10)))
    request.add_header('User-Agent', random.choice(headers_useragents
    ))
    request.add_header('Cache-Control', 'no-cache')
    request.add_header('Accept-Charset', 'ISO-8859-1,utf-8;q=0.7,*;q
    =0.7')
    request.add_header('Referer', random.choice(headers_referers) +
        buildblock(random.randint(5,10)))
    request.add_header('Keep-Alive', random.randint(110,120))
    request.add_header('Connection', 'keep-alive')
    request.add_header('Host',host)

```

Codigo Python 2.1: Fragmento de código fuente de Hulk

Siege [10] es también otra herramienta que permite realizar pruebas de rendimientos en servidores web es utilizada en ataques de denegación de servicio. Siege normalmente se emplea para dimensionar las capacidades de un servidor web, pero claramente tiene otros usos para los cuales no fue diseñada.

Finalmente, mencionamos *Low orbit ion cannon* (Loic) [11, 12], que ha sido la herramienta principal que ha estado empleando el grupo Anonymous para la realización de sus ataques. Existen diversas variaciones de Loic, como puede ser Hoic y otra versión que esta embebida en java script, ambas utilizadas para el mismo fin.

Se mencionan a continuación noticias sobre los últimos ataques producidos en Internet, que ponen de manifiesto el continuo crecimiento de los mismos, especialmente en términos de duración y de complejidad.

Un artículo del año 2008 de Arbor Networks [13], indicaba que los ataques DDoS continuarían creciendo en tamaño y en sofisticación. El más grande observado en el 2008 fueron de 40 Gbps, aproximadamente un 67% más respecto del año anterior.

En abril del 2009, se produjeron múltiples ataques a los servidores DNS con grandes cantidades de resoluciones [14]. En mayo del mismo año un ataque DDoS a los servidores DNSs de China provocó, que durante varias horas, no funcionase el servicio DNS en China [15].

El 20 de marzo del 2010, una organización criminal registró una serie de dominios para la implantación de una nueva Botnet que permitiera realizar ataques DDoS previo pago a la organización [16]. Este servicio fue de dominio público y estuvo hospedado en China, donde con un simple ingreso en una cuenta bancaria y la inclusión del dominio a atacar se disponía de miles de maquinas para realizar efectivamente el DDoS.

El 25 de Mayo del 2010, uno de los mayores hostings americanos, Media

Temple (www.mediatemple.net) se vio afectado por uno de los mayores ataques sufridos en sus servidores DNS. De acuerdo con el informe que publicó la compañía [17], el ataque duro aproximadamente dos horas y 5 minutos y provocó la interrupción de los servicios de clientes como Adobe, ABC, Sony, NBC, Time, Volkswagen y Starbucks.

El 28 de noviembre de 2010, la infraestructura de Wikileaks fue atacada con una intrusión DDoS [18, 19] como publicaron en su cuenta de Twitter. Con este tipo de ataque se consiguió que muchos de los usuarios que accedían al portal Wikileaks no pudieron acceder a su página principal.

La compañía Arbor Networks [13] informó que justo después del ataque que sufrió Wikileaks, todo el tráfico fue redirigido desde su proveedor 'Cablegate', un *Internet Service Provider* (ISP) Sueco, a sus servidores respaldo en Francia y Estados Unidos que tenían copias exactas del servidor original de Wikileaks

En respuesta a los ataques que sufrió Wikileaks, el 1 de diciembre de 2010 [20, 21] el grupo Anonymous realizó un ataque masivo DDoS contra Mastercard por no permitir las donaciones gratuitas a Wikileaks. La herramienta que empleó Anonymous para realizar dicho ataque fue Loic.

Prolexic [22] comenta que comparando el último cuarto del 2012 con respecto a 2011, se han incrementado el número de ataques DDoS en un 88 %. También se ha reducido la duración de los ataques de media de 33 horas a 19 horas y se ha incrementado en un 230 % los ataques que saturan el ancho de banda.

Actualmente, la mayoría de los ataques DDoS se realizan mediante TCP (90 %-94 %) [23], para ser exactos mediante HTTP, que es el protocolo más predominante de Internet.

A continuación, en el apartado 2.2 del presente capítulo, detallamos la anatomía de un ataque DDoS. Seguidamente, presentamos los modelos de prevención en la sección 2.3. Después, en el apartado 2.4 incluimos una introducción a las técnicas de detección de ciberataques y una clasificación de los trabajos mas relevantes. En la sección 2.5 exponemos diversas técnicas de comunicación entre sistemas de manera segura y, finalmente, en el apartado 2.6 presentamos la clasificación y explicación de las técnicas más relevantes para mitigar el impacto de un ataque DDoS.

2.2. Anatomía de un ataque

Actualmente existen dos grupos organizados y distribuidos a los que se les atribuyen la mayoría de los ataques de denegación de servicio, Anonymous y LuzSec, que son los más activos hasta el año 2013 [24, 25]. El modo de actuación de Anonymous consiste en intentar mediante inyección de código SQL [26, 27] robar información de bases de datos y, en caso de no poder conseguirla, sincroniza miles de Hosts y lanza un ataque de denegación de servicio a nivel de aplicación generalmente.

Anonymous suele utilizar herramientas DDoS [28] como Loic, Hoic, Slowloris entre otras. Mediante dichas herramientas se consiguen generar millones de flujos para denegar el servicio en infraestructuras web.

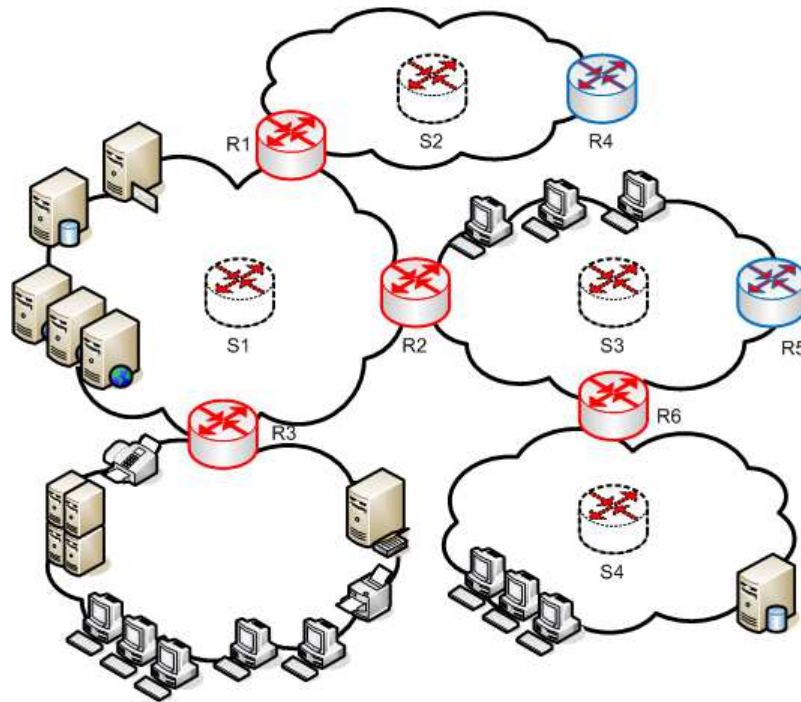


Figura 2.1: Infraestructura de red

En la figura 2.1 se muestra una infraestructura de red que dispone de los siguientes elementos:

- Dos Routers frontera R4 y R5 de acceso al exterior, a un mismo ISP o diferentes según requisitos de red.
- Cuatro Routers/Switchs R1,R2,R3 y R6 internos que gestionan los diferentes segmentos de red, en este caso dispondrían de 5 segmentos de red.
- Servidores, equipos de trabajo, impresoras y cualquier otro elemento susceptible de ser conectado a una red TCP/IP.

Dependiendo del grado de complejidad del ciberataque pueden producirse los siguientes tipos de intrusiones:

- Exteriormente desde fuera de la red vía DDoS, este ataque se realiza mediante Botnets que pueden estar en cualquier lugar del mundo, de modo que los flujos entrarían por los Routers de acceso R4 y R5.
- Desde fuera de la red vía DDoS, pero con supervisión interna del estado del ataque y mediante acceso a uno de los elementos exteriores como pueden ser R1 o R6 y con los flujos llegando por los Routers de acceso.

- Dentro de la red, desde cualquier elemento direccionable vía IP, de manera que los flujos de datos se generen desde cualquiera de las subredes internas.

Con la finalidad de proceder con el ataque, los agresores realizarán un estudio de los servicios a atacar y determinarán si el ciberataque se realizará interna o externamente, así como el modo de materialización de los mismos, en la mayoría de las intromisiones DDoS éstas son realizadas mediante el protocolo HTTP. Posteriormente, una vez definido el objetivo, mediante el empleo de Botnets [29, 30] los agresores sincronizarán miles de ordenadores para lanzar peticiones HTTP al objetivo. De esta forma, el objetivo recibirá tantas peticiones que, dependiendo del nivel de complejidad y de seguridad de su red, o bien se saturará o bien la agresión no surgirá efecto. En términos generales, la duración de los referidos ataques suele ser de aproximadamente dos horas de manera que, una vez el ataque cese, teóricamente los servicios volverán a restablecerse.

Motivaciones en los ataques DDoS Cabe mencionar que son muchas las causas que motivan la materialización de un ataque DDoS. Sin embargo, todos ellos cuentan con un objetivo común que consiste en la deshabilitación de un servicio de Internet, de manera que el objetivo atacado no pueda continuar prestando el servicio para el que fue diseñado. En términos generales este tipo de ataques DDoS son llevados a cabo tanto por organizaciones distribuidas, entre las que se incluyen Anonymous o Luzsec, así como por otras de tipo gubernamental. Y con todo ello, y en base al marco descrito, compañías rivales, organizaciones criminales, grupos de hacktivistas, organizaciones gubernamentales (NSA, MI5) y otros, llevan a cabo dichas agresiones con la finalidad de interrumpir el funcionamiento de infraestructuras web, centrales nucleares, centralitas telefónicas, servicios de pago, etc...

En la figura 2.2 podemos observar como los últimos ataques producidos en octubre de 2013 fueron de índole criminal, con respecto a otro tipo de motivaciones. No obstante esta tendencia puede ir variando en base de las motivaciones políticas, económicas y sociales que imperen en ese instante.

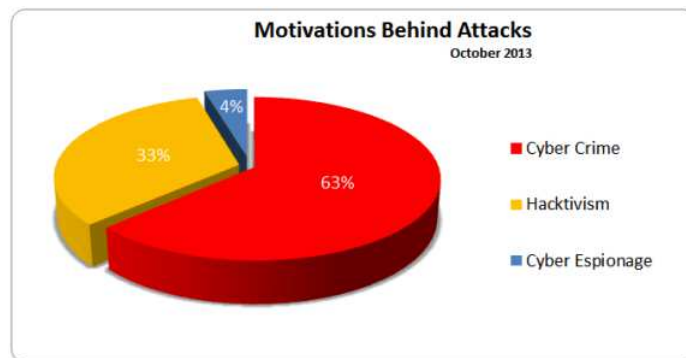


Figura 2.2: Motivaciones de ataques octubre 2013 [31]

2.3. Modelos de prevención

Existen múltiples modelos de prevención y contra-medidas que tienen por objeto la mitigación de este tipo de ataques. En el caso que nos ocupa, con la finalidad poder enfocar las diferentes técnicas de mitigación, podemos basarnos en el modelo de Mirkovic *et al* [32] (ver figura 2.3).

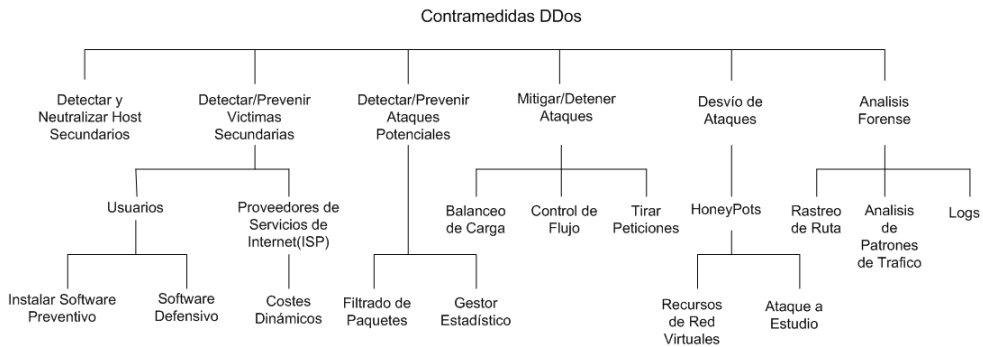


Figura 2.3: Modelo de Mirkovic

Este modelo clasifica en seis grupos las medidas y contramedidas a adoptar para prevenir los ataques. Entre ellas se incluyen las actuaciones para la detección de los host secundarios, para la detección de las víctimas indirectas, para la prevención de los potenciales ataques, para la mitigación de los mismos, para su desvío y, finalmente, para su análisis forense. Esta clasificación constituye un modelo de referencia para detectar, analizar y resolver los problemas derivados de este tipo de ataques.

En los siguientes subsecciones detallamos cada uno de los grupos considerados en su modelo.

2.3.1. Detección y neutralización de Host secundarios

En este tipo de contramedida resulta esencial conocer perfectamente el funcionamiento de la herramienta y, mediante sistemas de detección de intrusos, disponer de reglas que permitan su detección. Muchas de estas herramientas cifran las comunicaciones para no ser identificadas, lo que dificulta todavía más su detección. Sin embargo, un buen análisis de dicha herramienta, así como de los tipos de flujos que ésta genera, nos ayudará a su futura identificación.

Para la detección de los Host secundarios, situados en la capa intermedia, es necesario disponer de un conocimiento adicional sobre cuales son los comandos, protocolos, etc... que utiliza la herramienta en cuestión. En este sentido, y dado que las demás herramientas emplean la misma filosofía, se procede con el análisis y la descripción del modelo de Trin00. En este modelo la comunicación se establece a través de los distintos niveles, según se describe en la figura 2.4.

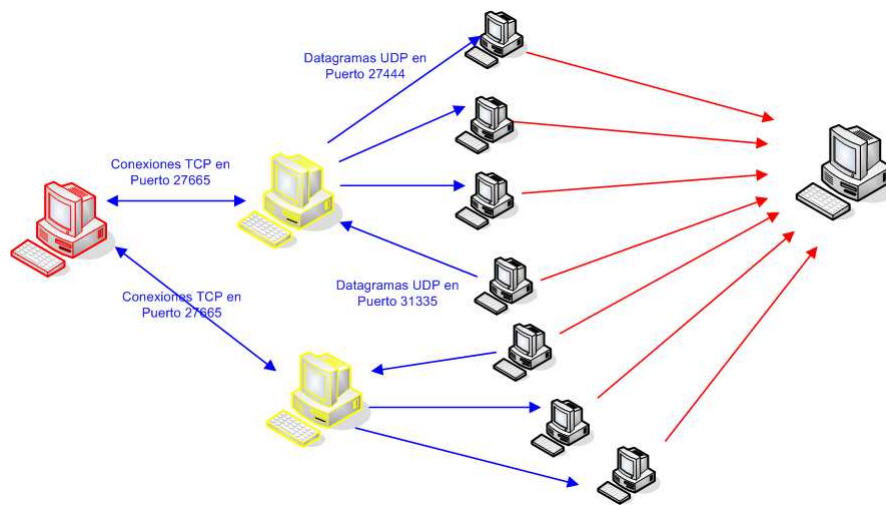


Figura 2.4: Arquitectura Trin00

Trin00 emplea los siguientes puertos para comunicar los diferentes procesos y demonios (proceso en segundo plano).

- Atacante a Host Secundario: 27665/TCP.
- Host Secundario a Demonio: 27444/UDP.
- Demonio a Host Secundario: 31335/UDP.

El control de la red Trin00 se realiza sobre el puerto 27665 TCP. Para ello, en primer lugar, se realiza una comprobación mediante una password, *betaal-mostdone*, que permite autenticar a los atacantes y, en caso de existir varios atacantes, envía un aviso a uno de ellos. El envío de los datagramas remitidos desde el Host secundario a los demonios se realiza por medio del puerto 27444, y estos comandos están separados por blancos y con la cadena "l44adsl", en la cual la subcadena "l44" nos indica que los comandos que sigan a continuación serán procesados.

Actualmente muchas de las técnicas basadas en la detección de hosts secundarios no son validas ya que el la utilización de canales encubiertos o de cifrado de comunicaciones dificulta la identificación de los mismos [33].

2.3.2. Detección y prevención de víctimas

Los principales afectados por los ataques de denegación de servicio son los ISPs, las *Content Delivery Networks* (CDNs) y las infraestructuras de las empresas, como pueden ser servidores web, correo, bases de datos, etc... Así mismo, a nivel de usuario resulta de gran importancia la concienciación del problema de denegación de servicio. En este sentido, con la finalidad de garantizar una

adecuada protección y evitar que el interesado sea objeto de ataques sin su conocimiento, resulta suficiente con el empleo de actualizaciones de software de seguridad y de Firewalls de uso personal.

Mankins *et al* [34] proponen un interesante modelo para que los ISPs puedan mitigar el impacto de los ataques DDoS. Dicho modelo está basado en un patrón económico que requiere que los solicitantes paguen por el empleo de recursos tales como la memoria y el uso del procesador. El modelo propuesto presenta las siguientes propiedades:

- El precio de un recurso es mayor si la disponibilidad de dicho recurso es menor. A menor disponibilidad de un recurso, mayor precio.
- El precio de un recurso es mayor para él que lo utiliza para un largo alcance. A mayor alcance del recurso, mayor precio.
- El precio de un recurso es inferior para los clientes de gran prioridad. A mayor prioridad de los clientes, menor precio.

En este modelo se despliegan unos agentes, tanto en el cliente como en el servidor, que interactúan entre sí. Los agentes determinan automáticamente el precio de cualquier servicio en base al estado del sistema, así como el modelo económico elegido. En ella se estudian diferentes modelos económicos y sistemas de pago para resolver el ataque al servidor mediante avalancha y el ataque al servidor por drenado. Esta investigación es una novedosa propuesta para los problemas de DoS, aunque sólo están disponibles los resultados preliminares. No obstante este modelo resulta difícilmente implantable sobre todo en España ya, al menos por ahora, los ISPs no están dispuestos a realizar una inversión de gran capital que permita implantar dicho modelo, y prefieren limitarse únicamente a gestionar tráfico.

En cualquier caso, pretender controlar la totalidad del flujo, desde su origen hasta el destino de la comunicación, resulta prácticamente imposible, tanto en términos de coste como técnicamente hablando. No obstante, el modelo de Mankins intenta aportar una solución teórica, aunque poco realista, relativa a la implementación y el despliegue.

2.3.3. Detección y prevención de ataques potenciales

Se exponen, a continuación, dos tipos de técnicas para la prevención y detección de ataques. Ambas incorporan tablas de estado de los paquetes IP que les permiten descartar paquetes en base a ciertos criterios previamente establecidos.

Filtrado de paquetes distribuidos basado en la ruta

El filtrado de paquetes distribuidos basados en la ruta (*Distributed Packet Forwarding*, DPF), propuesto por Park *et al* [35], constituye una técnica para prevenir ataques DDoS utilizando direcciones IP falseadas en el origen. Esta técnica generaliza el filtrado de acceso, que normalmente únicamente se realiza en los bordes de la red, de manera que Internet puede entenderse como un

conjunto de sistemas autónomos independientes. El DPF se instala en cualquier conexión entre dos sistemas autónomos independientes. Un paquete IP con origen x y destino y que circula a través de un determinado enlace puede eliminarse si dicho enlace no pertenece a cualquier ruta conectada a x e y . Se resumen a continuación los principales objetivos del DPF:

- Maximizar el filtrado de direcciones IP falseadas.
- En el caso de paquetes falsos que superen el proceso de filtrado, reducir el número de sitios que podrían haber enviado los paquetes (con el objeto de facilitar la traza hacia atrás).
- Reducir el número de Routers en los que se realiza el filtrado basado en el enrutamiento.
- Identificar los sitios óptimos en los que se debería realizar el filtrado.

Filtrado del acceso

Mediante el filtrado de paquetes seleccionados es posible disminuir considerablemente el impacto de algunos ataques DDoS o incluso, en algunos casos, bloquearlos. Por ejemplo, el filtrado de paquetes en el borde de la red debe prevenir el flujo de paquetes IP falsos desde Internet, que tiene por objeto atacar un ordenador principal en una red de área local (*Local Area Network*, LAN), o de un atacante en una LAN que ataca a un ordenador principal en Internet pretendiendo ser una persona diferente (engaño del origen).

Esta técnica requiere que todos los paquetes entrantes deberían filtrarse con la finalidad de descartar paquetes falsos tan rápido como sea posible. Este proceso, conocido como filtrado del acceso [36], se puede basar en los campos de la cabecera del protocolo IP, ICMP, UDP y TCP, a continuación vamos a ver como se desarrolla dicha técnica.

La Técnica de filtrado de acceso, también denominada *Unicast Reverse Path Forwarding* (uRPF) [37], permite el filtrado de acuerdo con la norma RFC 2827 [36]. uRPF permite descartar paquetes con direcciones falseadas en los Routers fronteras dependiendo del método usado. Para ello, resulta conveniente recordar que uRPF solo funciona en el interfaz de entrada del Router frontera.

uRPF se puede implementar de dos formas, en modo estricto y en modo pérdida. En modo estricto el Router toma la dirección origen del paquete y la valida contra la *Forwarding Information Base* (FIB) del interfaz de entrada del Router, seguidamente verifica que existe una ruta a la dirección origen en la tabla de adyacencia del interfaz, y si no existe entonces el paquete se descarta. El modo pérdida fue diseñado para entornos frontera entre ISPs y éste únicamente realiza la comprobación de la FIB, sin realizar la otra comprobación.

Las técnicas detalladas suponen un importante avance en la prevención de ciberataques. Por otra parte, el despliegue de éstas ha sido prácticamente nulo ya que la efectividad no ha sido comprobada exhaustivamente.

2.3.4. Mitigación y detención de ataques

En este apartado citamos dos técnicas para la mitigación y detención de los ataques DDoS que se basan en el control de los flujos de red responsables del ataque, medidas que generalmente suelen adoptarse sobre la capa de red TCP/IP.

Control de flujo

Uno de los mecanismos más interesantes para mitigar, e incluso detener, el ataque son los sistemas basados en control de flujo. Estos sistemas detectan y notifican una posible congestión de tráfico 'malo' y adoptan las medidas necesarias para solucionarlo. La técnica propuesta por Ioannidis *et al* [38], denominada Pushback, constituye una de las primeras propuestas para tratar este problema desde la perspectiva de la red. Este método generaliza la propuesta de limitación de velocidad utilizada en los métodos de calidad de servicio (*Quality of Service*, QoS), que no diferencia entre flujos atacantes y no atacantes, si no que se basa en ciertos parámetros de congestión. Cada Router de la red, implementado con OpenBsd, está compuesto por los siguientes elementos (ver figura 2.5):

- Una cola de salida gestionada mediante un mecanismo de descarte capaz de evitar desbordamientos.
- Un agente añadido, llamado *pushbackd*, para el análisis de los paquetes eliminados por el mecanismo de descarte que permite identificar algunas firmas.
- Un módulo que limita la velocidad para controlar el número de paquetes que se corresponden con las firmas atacantes que entran en la cola de salida.

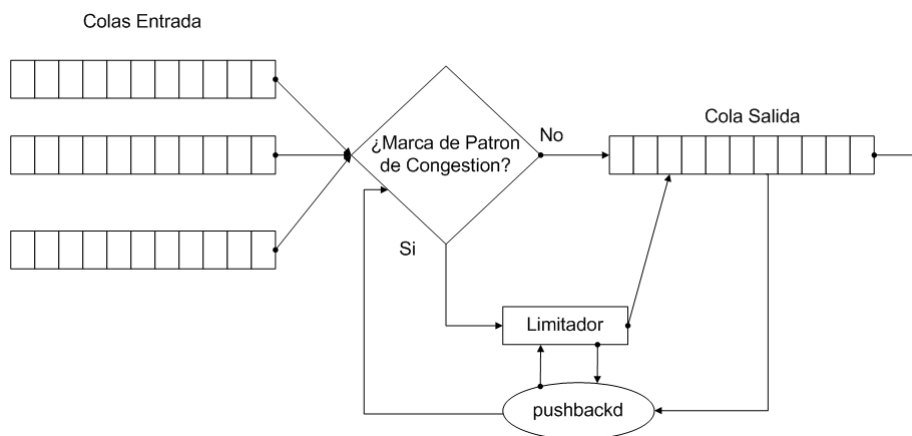


Figura 2.5: Arquitectura Pushback

Si un elemento de red no detecta un ataque, los paquetes entrantes se encaminan directamente a la cola de salida. Tan pronto comienza el mecanismo de descarte de la cola para eliminar paquetes, el agente añadido intenta identificar secuencias. En el caso de identificar una secuencia, cualquiera de los siguientes paquetes entrantes que se ajusten a la misma deberán ser transmitidos al módulo que limita la velocidad antes de posicionarlo en la adecuada cola de salida. Una vez se ha identificado que un elemento de red ha reconocido una secuencia atacante, éste reacciona localmente y, a continuación, propaga la reacción a los Routers ascendentes, tal y como puede observarse en la figura 2.6.

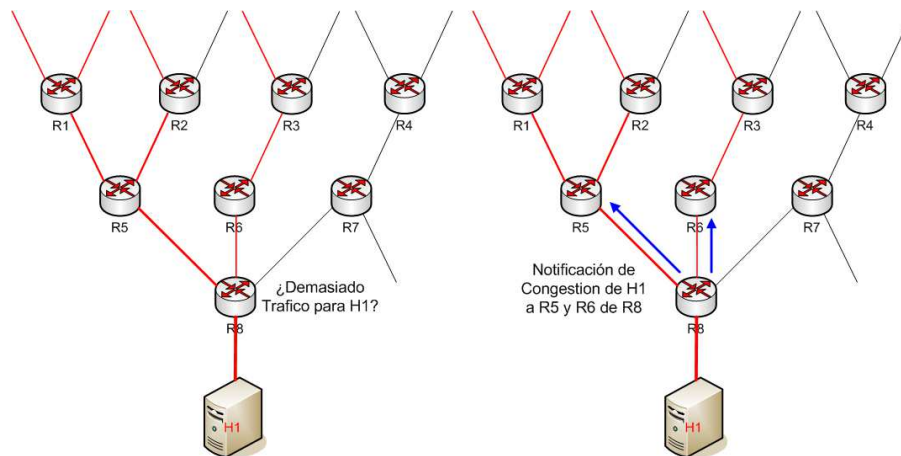


Figura 2.6: Notificación Pushback

Con la finalidad de mitigar el posible ataque, en cada uno de los Routers que reciben la información de congestión, se aplicarán limitaciones de ancho de banda a los flujos implicados en el destino de la agresión que, en el caso de la figura 2.6, correspondería al equipo H1.

Calidad de servicio

La calidad de servicio, también denominada *Quality of Service* (QoS), hace referencia a un rango entero de técnicas [39] que permiten a un sistema abierto mejorar la calidad de servicio de los flujos seleccionados en detrimento de algunos otros flujos.

Las técnicas propuestas constituyen un amplio campo de investigación en interconexión y telecomunicaciones y, de hecho, algunas de las técnicas QoS pueden ser empleadas para disminuir la eficacia de los ataques DoS, especialmente de aquellos ataques distribuidos basados en el agotamiento del ancho de banda. QoS incluye limitación de velocidad, asignación de prioridades y re-clasificación de cualquier flujo de tráfico, en especial en situaciones de ataque.

Eligiendo algoritmos de encolamiento y políticas QoS adecuadas es posible disminuir el impacto de los ataques DoS.

Dentro de las técnicas que se basan en QoS se encuentra la de Garg [40]. Su implementación está basada en la utilización de un clasificador de paquetes de red que, en base a los recursos disponibles, asigna a los diferentes flujos diversas clases de tráfico. Estas últimas pueden ser controladas bien por control de ventana o bien por limitación de velocidad (*Rate Limiting*) y la otra clase se encuentra regulada por control de ventana. Por otra parte, el sistema mantiene un control del estado del servidor atacado de manera que pueden ajustarse los valores de las clases de tráfico.

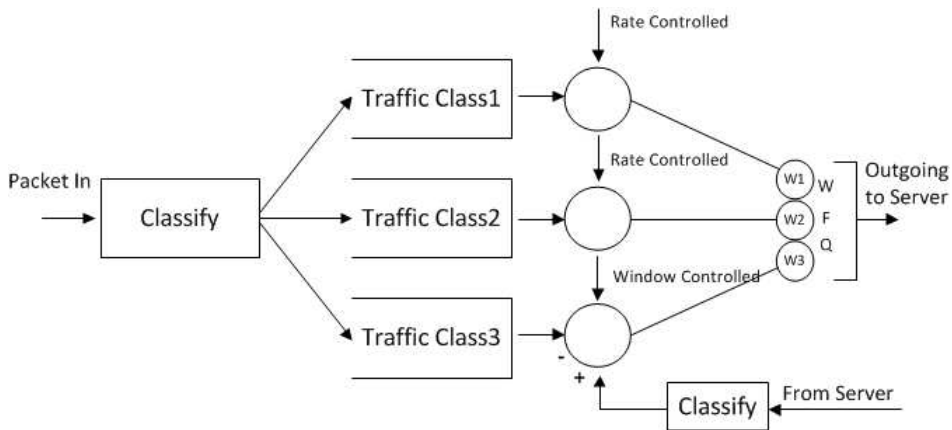


Figura 2.7: Regulación QoS [40]

Si bien en la actualidad las técnicas basadas en control de flujo son las más efectivas, la evolución de los ataques durante los dos últimos años pone de manifiesto el cambio de tendencia en la utilización del protocolo TCP, que está siendo sustituido por el protocolo HTTP, lo que refleja que a día de hoy dichas medidas están obsoletas.

2.3.5. Desvío de ataques

Una de las técnicas más novedosas de reciente aparición son las conocidas HoneyPots [41, 42]. Su principal finalidad consiste en que éstas deben ser comprometidas como recursos de una red, de manera que, con la finalidad de ganar tiempo y poder mitigar el ataque, éstas sean más susceptibles a ser atacadas.

El primer Honeypot de la historia fue Labrea [43], que lograba mitigar los ataques del virus CodeRed en el momento de su migración en una LAN a otro huésped mediante el mecanismo de ventana deslizante de TCP.

Con ellos, en caso de que los Honeypots se vean comprometidos para formar parte de un ataque DDoS, se consigue reducir drásticamente el árbol de

expansión que se empleará en dicho ataque, tal y como se muestra en la figura 2.8.

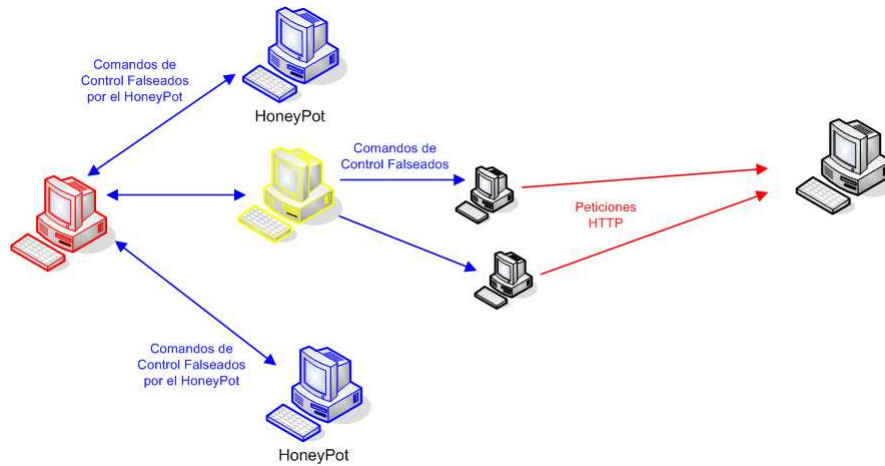


Figura 2.8: HoneyPots

Los HoneyPots podrán enviar información falsa al atacante, de manera que éste creará que tiene más Hosts para su utilización en el ataque. En ocasiones resulta interesante aprender de los propios atacantes. Para ello, en las redes se ubican servidores dispuestos a conciencia para que los intrusos los saboteen y son monitorizados por sistemas que actúan como puentes a los servidores, registrando de forma transparente los paquetes que acceden a dichos servidores.

De esta forma, una vez detectado un ataque (por modificación de la estructura de archivos o por la utilización de las llamadas al sistema), se recompone la traza del atacante (secuencia de paquetes registrados en el monitor puente) y se realiza un análisis forense que, en caso de identificar un nuevo ataque, concluye en una nueva regla de detección que será transmitida a un sistema de detección de intrusos.

Provos [44] diseña e implementa Honeyd, que consiste en un framework de HoneyPots virtuales (como podemos ver en la figura 2.9) que simulan sistemas a nivel de red. Honeyd soporta funciones a nivel IP, responde a servicios tal y como si éstos existieran dentro de la red para cada HoneyPot diferente y también ofrece funciones para simular diferentes pilas TCP/IP de sistemas operativos, ver figura 2.10.

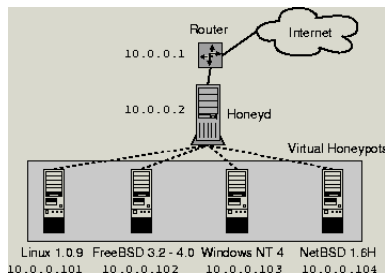


Figura 2.9: Arquitectura de honeyd [44]

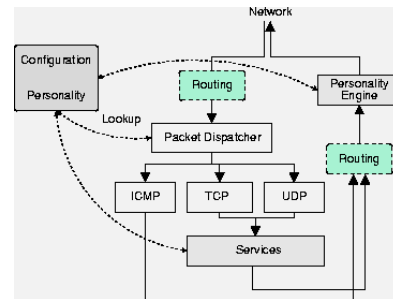


Figura 2.10: Funcionamiento y diseño de honeyd [44]

Khattab *et al.* [45] proponen un mecanismo de detección y mitigación de DoS a nivel de servicio basado en Honeypots en continuo movimiento que impiden determinar su localización. Para la gestión de los Honeypots, se realizan túneles IP de manera que un servidor central administra la movilidad de éstos. Esta técnica es novedosa en términos de funcionalidad, sin embargo es poco efectiva en términos de seguridad, ya que la gestión de los túneles IP se lleva a cabo en un servidor susceptible de ser atacado y, como consecuencia de ello, dichos túneles IP pueden ser atacados o deshabilitados.

Los Honeypots constituyen una nueva vía de exploración e investigación. Sin embargo, éstos suelen ser más utilizados para recopilación de datos y comportamiento de usuarios maliciosos que para la mitigación o detención de un ataque DDoS de manera directa.

2.3.6. Análisis forense

El análisis forense [46, 47] de sistemas consiste en el proceso de extracción, conservación, identificación, documentación, interpretación y presentación de las evidencias digitales, de forma que éstas puedan ser aceptadas en cualquier proceso legal, como por ejemplo un juicio. Dicho análisis proporciona técnicas y principios que facilitan la investigación del delito y su metodología básica consiste en:

1. Adquirir las evidencias sin alterar ni dañar el material original. La forma ideal de examinar un sistema consiste en detenerlo y examinar una copia de los datos originales. No obstante, conviene resaltar que no se puede examinar un sistema presuntamente comprometido utilizando las herramientas que se encuentran en dicho sistema pues éstas pueden estar afectadas por un virus o rootkit. La cadena de custodia documenta el proceso completo de las evidencias durante la vida del caso, quién la recogió y dónde, quién y cómo la almacenó, quién la procesó etc. Cada evidencia deberá ser identificada y etiquetada, a ser posible en presencia de testigos, con el número del caso, una breve descripción, la firma y la fecha en que fue recogida.

2. Comprobar (Autenticar) que las evidencias recogidas, y que van a constituir la base de la investigación son idénticas a las abandonadas por el delincuente en la escena del crimen. Para ello, resultan de gran ayuda las técnicas y herramientas de control de integridad que mediante la utilización de una función hash generan una huella electrónica digital de un fichero o un disco completo.
3. Analizar los datos sin modificarlos. En este punto, resulta crucial proteger las evidencias físicas originales. Para ello resulta conveniente realizar dos copias de los originales, de modo que en caso de error se pueda recuperar la imagen original y continuar con el análisis de forma correcta. Estas copias deben ser clones realizados bit a bit del dispositivo original, los backups normales no copian ficheros que han sido borrados ni determinadas partes de los discos que pueden contener pistas importantes para el desarrollo de la investigación. Para ello, resulta esencial realizar siempre un control de integridad de la copia realizada antes de comenzar ningún análisis.

El análisis forense en DDoS no suele ser muy utilizado ya que los ataques DDoS dejan poco rastro si lo comparamos con otro tipos de intrusiones. Las evidencias y los datos resultantes de un ataque suelen ser los logs de los sistemas atacados y éstos se utilizarán para el análisis de la intrusión.

2.4. Modelos de detección

Durante los últimos años se han propuesto muchas técnicas que permiten detectar los ataques DDoS [48, 49, 50, 51, 52]. A continuación, se exponen algunos de los diversos métodos propuestos por diferentes autores.

Para la detección de ataques Mirkovic *et al.* [53], utilizan métricas basadas en la experiencia de los usuarios de QoS de forma que, mediante el empleo de dichas métricas de tráfico, pueden definirse los límites de lo que sería un ataque frente a lo que no.

El método propuesto por Cavrilis *et al.* [54] se basa en la utilización de links señuelos situados en las páginas web del servidor. Estos links son invisibles para un humano, pero no para un Bot de forma que cuando se accede a ellos se envía una alarma. Los resultados que nos muestran en servidores reales les da un valor de falsos positivos muy pequeño, aproximadamente un 0.0421 %, sin embargo su sistema de detección tiene que alterar las páginas web del servidor que sería la parte negativa de su aproximación.

Stefan *et al.* [55] analizan la efectividad del empleo de técnicas de aprendizaje automático para detectar ataques. Para ello, entran en un proceso de captura de tráfico y, posteriormente, utilizan una red neuronal como sistema de detección. Como se observa en la figura 2.11, las probes envían datos a un servidor central. Este servidor correlacionará los datos recibidos para generar filtros y ayudas en la gestión de la detección del ataque.

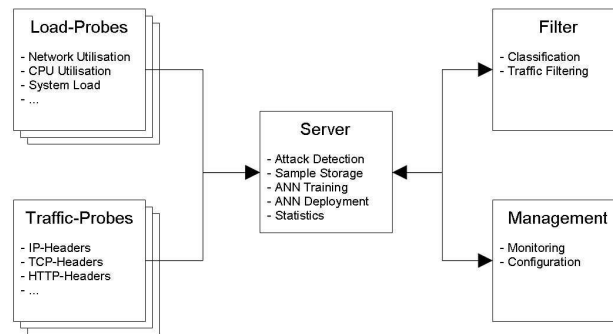


Figura 2.11: Arquitectura propuesta por Stefan [55]

Mediante el empleo del sistema de detección de intrusos Bro [56, 57], Callahan *et al.* [58] recolectan como es el tráfico web, incluyendo información relativa al número de transacciones, tipos, tamaños, así como funcionamiento las redes de entrega de contenidos (CDNs), entre otros aspectos.

Chonka *et al.* [59] proponen el empleo de la teoría del caos para detectar diferencias en lo que es tráfico legítimo y el que no lo es. Observan que el tráfico de un ataque sigue un patrón de tráfico, de esta forma desarrollan un algoritmo que detecta ataques y que dan también una aproximación de cuando comenzó este.

Li *et al.* [60] utiliza la distribución de energía basado en wavelets para detectar ataques. El análisis de wavelets permite hacer complicadas correlaciones de series continuas de tiempo con un bajo coste computacional.

Danner *et al.* [61] presentan un algoritmo de complejidad $O(n)$ para detectar ataques en la red Tor [62]. Esta red es actualmente una de las redes más famosas de anonimización en Internet. Es una red de túneles virtuales que permite a la gente y a grupos de gente mantener la seguridad y privacidad en Internet. También permite a los desarrolladores de software crear nuevos sistemas de comunicación que protejan la privacidad.

Wang *et al.* [63] proponen una arquitectura llamada 'Patricia' en la que las redes cooperan entre ellas para controlar los canales de datos y control. Su arquitectura está basada en tres pilares fundamentales. En primer lugar proporcionan un mecanismo de cooperación entre redes para compartir información sobre usuarios sospechosos. A continuación, utilizan un procedimiento de aprobación entre redes, un nodo origen tiene que obtener la aprobación de su propia red y una autorización de la red destino. Y por último, un mecanismo de control del tráfico de manera que únicamente llegue a un Host solo cuando sea necesario.

Yatagai *et al.* [64] proponen dos algoritmos para detectar ataques basados en HTTP, uno centrado en la forma de acceso a las paginas web y el otro basándose en la correlación de tiempo de acceso entre las paginas y su tamaño. Este modelo es de los primeros que van centrandose en el protocolo HTTP usando para ello

la teoría de grafos.

Siaterlis *et al.* [65] proponen la utilización de diversas métricas para la detección de ataques en el borde de la red y su clasificación mediante una red neuronal. Para ello, realizan un aprendizaje supervisado en el que, mediante las estadísticas que toman, entrenan a la red neuronal multicapa preparada para ataques basados en tráfico UDP.

Seguidamente, en la tabla 2.1, hemos realizado una clasificación de los autores anteriores en función de las siguientes técnicas: Métricas L4 en caso de que el sistema utilice métricas a nivel de red (TCP/IP); Distribuido, si el sistema emplea algún tipo de distribución; Aprendizaje automático, si la técnica requiere algún tipo de aprendizaje supervisado o no supervisado; Frecuencias, si existe algún análisis frecuencial ya sea de tiempo o de bytes; Métricas L7, si utilizan métricas a nivel de aplicación y, finalmente, Redes neuronales, si emplean alguna técnica referente a redes de neuronas.

Autor	Métricas L4	Distribuido	Aprendizaje automático	Frecuencias	Métricas L7	Redes neuronales
Mirkovic [53]	*					
Cavrilis [54]					*	
Stefan [55]	*		*			*
Chonka [59]		*				*
Li [60]	*			*		
Danner [61]		*				
Wang [63]	*	*				
Yatagai [64]					*	
Siaterlis [65]	*					*
Campo [66]	*	*	*		*	

Tabla 2.1: Clasificación de las técnicas de detección

En esta tesis proponemos un sistema para la detección [66] que consiste en un algoritmo capaz de analizar los flujos HTTP que se dirigen a los servidores web en tiempo real. El análisis de dicho algoritmo combina métricas a nivel TCP/IP y HTTP, así como un aprendizaje de las peticiones más frecuentes y de los caminos más utilizados, todo ello combinado con la capacidad de distribuir el algoritmo en diferentes nodos.

Como bien es sabido, no existen sistemas de detección universales, que sean aplicables en todos los casos y que permitan poner de manifiesto la materialización de dichos ataques. Por este motivo, este campo de trabajo constituye un área de de gran interés para la investigación y el desarrollo.

2.5. Modelos de comunicación

La monitorización de sistemas ha sido estudiada desde hace tiempo y ha constituido una línea de investigación durante estos últimos años. Los sistemas de comunicación o monitorización seguros tienen la necesidad de compartir información, generalmente crítica, de manera segura para que ésta no pueda ser

alterada ni utilizada con otros fines. Por ejemplo, sistemas como DIMES [67], DShield [68] o NetBait [69], utilizan información sobre incidentes de seguridad y cooperan con otros repositorios mostrando los sucesos reportados por otros sistemas. Sin embargo, a pesar de los requerimientos de seguridad, los sistemas de monitorización se limitan a ser meros repositorios de información ya que únicamente cifran sus comunicaciones a la hora de enviar el reporte. Como consecuencia de ello los sistemas como Firewalls, *Network Intrusion Detection Systems* (NIDS) o Proxies quedan expuestos a los ataques ya que, para reportar al servidor, utilizan protocolos orientados a conexión, como puede ser TCP, que pueden ser rastreados por un atacante con la finalidad de conocer la topología existente.

Estos sistemas deben ser capaces de reportar incidentes y de enviar información crítica a través de las redes. Para ello se pueden emplear varias técnicas. La esteganografía [70, 71, 72, 73, 74], método frecuentemente utilizado para ocultar información, se basa en el empleo de imágenes en las que se insertan contenidos que nada tienen que ver con ellas de forma que, si bien la información a ocultar es visible, ésta no lo es a simple vista. Otra de las técnicas empleadas es la conocida como "Técnica de canales ocultos" (*Cover channels*) que oculta los contenidos en flujos de red [75, 76, 77, 74].

Estas técnicas generalmente se basan en camuflar la información en campos de protocolos, tales como IPv4/v6, TCP, DNS, HTTP, etc.... También existen aplicaciones comerciales como Skype [78], que utiliza canales encubiertos en tráfico HTTP para ocultar sus comunicaciones de VoIP y, de este modo, saltarse las restricciones que suelen poner los ISPs. A continuación se va a proceder con la descripción de los estudios más relevantes e interesantes relativos a la técnica de canales encubiertos, que se complementa con la tabla 2.2 de clasificación de las técnicas mencionadas.

Sebastian *et al.* [79] comparan las diferentes técnicas de codificación de información y proponen un mecanismo para codificar la información en el campo Time to live (TTL) del datagrama IP. Ellos agrupan las diferentes técnicas para ocultar la información en el TTL y lo categorizan en tres clases: (a) codificar directamente en los bits del campo TTL, (b) establecer una correspondencia entre valores típicos de TTL y valores específicos de TTL como pueden ser 256, 128 y 64 y (c) correlacionar diferentes TTLs de paquetes IP para codificar información. La principal debilidad de su modelo radica en la cantidad de datagramas IP que hay que enviar para codificar la información, así como la limitación de valores que se pueden codificar en el campo TTL de IP, que se establece en 8 bits.

Lucas *et al.* [80] proponen un sistema denominado TUNS que se basa en ocultar la información en el protocolo DNS. Su sistema solamente utiliza el campo CNAME de la cabecera DNS. Utiliza una codificación base 32 sin necesidad de fragmentar la DNS.

Xiapu *et al.* [81] implementan un sistema llamado CLACK, donde codifican los mensajes en los acuse de recibo de TCP. Su sistema se basa en la persistencia de los flujos TCP buscando dos objetivos: aportar fiabilidad al canal al igual que se haría en un flujo TCP normal e incrementar el coste de detectar el canal

oculto.

Sanders *et al.* [82] proponen un sistema basado en el concepto de entropía y de aprendizaje. Su modelo se basa en el cálculo de la entropía de ciertos campos de diferentes protocolos como pueden ser ICMP, ICMPv6, IP, etc... Aunque su estudio carece de resultados fiables en términos de falsos positivos y negativos, su modelo aplica el concepto de aprendizaje y de análisis en el campo de los canales encubiertos. Como podemos observar en la figura 2.12, el flujo de datos que sigue su propuesta es cíclico. Así mismo, cabe destacar que el sistema aprende mediante el cálculo de la entropía de los flujos de red.

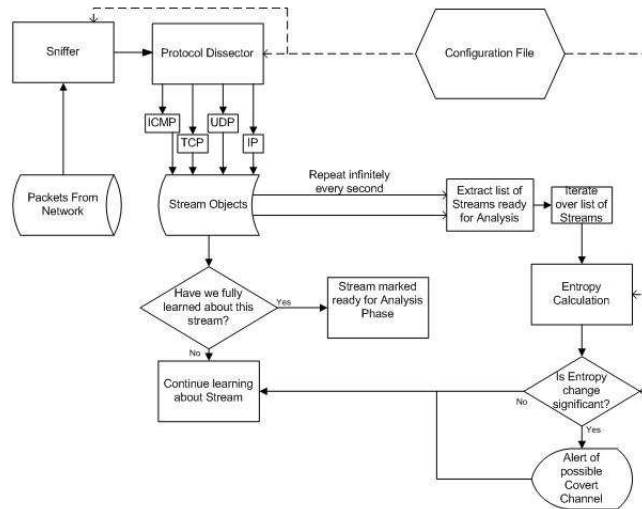


Figura 2.12: Prueba de concepto de Sanders [82]

Szczypiorski *et al.* [83] proponen *Hidden Communication system for Corrupted networks* (HICCUPS), un sistema esteganográfico para redes medianas incluyendo redes inalámbricas locales. HICCUPS emplea comunicaciones seguras y otros métodos criptográficos para la implementación de una solución completa y proponen un nuevo protocolo para entornos inalámbricos que se basa en emplear tramas corruptas en el momento de solicitar ancho de banda.

Wojciech *et al.* [84] presentan un método de ocultación en paquetes fragmentados IP. Básicamente modifican el valor del offset del paquete fragmentado IP para añadir información. La principal novedad del sistema radica en la utilización de la fragmentación IP para el envío de la información. Sin embargo, al ser la fragmentación de paquetes IP fácilmente detectable, su empleo debilita considerablemente el sistema y conlleva grandes problemas en las redes, motivo por el cual la mayoría de estas la evitan.

Lucena *et al.* [85] proponen un canal de ocultación basado en protocolos de nivel de aplicación y son capaces de ocultar mensajes en protocolos, como

pueden ser SSH y HTTP, sin alterar la semántica del mismo y dificultando así su detección.

Akleyek *et al.* [86] presentan un planteamiento en esteganografía para sistemas *Peer to Peer* (P2P). De forma que emplean la infraestructura de una red P2P para ocultar la información. Este artículo sirvió de referencia para el modelo de comunicación que proponemos y que se describe más adelante.

Xinwen *et al.* [87] muestran una arquitectura basada en flujos para camuflar la información. Su propuesta consiste en incorporar paquetes en los que se incluye el mensaje original entre los paquetes de otro flujo, sobre servicios que van a puertos conocidos.

Jana *et al.* [88] estudian las aplicaciones VoIP para poder extender algoritmos esteganográficos. Para ello, utilizan parte de información del audio (PCM) para ocultar en estos campos la información. En su trabajo muestran resultados sobre su framework de los que se desprende que son capaces de ocultar la información durante los primeros 13 segundos.

Yali *et al.* [89] utilizan canales encubiertos basados en el tiempo y codifican el mensaje en el retardo entre paquetes, de forma que emulan las características del tráfico normal. Su sistema está diseñado para funcionar sobre tráfico UDP y, más concretamente, en tráfico que generan los juegos on-line.

Cole [74] presenta Stego, un framework completo para ocultar información que combina diferentes técnicas, como pueden ser esteganografía, criptografía, canales ocultos, etc... Para ello utiliza una aplicación estándar de Graig Rowlan [90] que le permite ocultar información en los campos ID del datagrama IP y en los números de secuencia de TCP.

A continuación, en la tabla 2.2, se clasifican las diferentes técnicas de canales de ocultación en base a los siguientes criterios: Cifrado, en caso de que el sistema utilice alguna técnica criptográfica; Modificación Layer 7 (L7) a nivel de aplicación, si el sistema modifica cabeceras como HTTP, DNS, etc... (de nivel de aplicación); Modificación IP, si la técnica implica modificar campos del datagrama IP para ocultar información; Modificación TCP/UDP, si existe alteración de cambios en las cabeceras TCP o UDP; Tiempo, si utilizan la variable tiempo para codificar los mensajes de alguna manera y, finalmente, Reactivos a la red, si los sistemas reaccionan según se comporte el tráfico de la red.

Autor	Cifrado	Modificación L7	Modificación IP	Modificación TCP/UDP	Tiempo	Reactivos a la red
Sebastian [79]			*		*	
Xiapu [81]				*	*	
Szczypiorski [83]	*					
Lucena [85]		*				*
Akleylek [86]	*			*		
Xinwen [87]		*	*	*		
Cole [74]		*	*	*		
Jana [88]		*			*	
Wojciech [84]			*			
Lucas [80]	*	*				
Yali [89]					*	
InCC [91]	*	*		*		*

Tabla 2.2: Técnicas de canales encubiertos

Como puede observarse en la tabla 2.2, la mayor parte de la clasificación está basada en almacenamiento o en canales de tiempo. Así mismo, gran parte de los estudios se fundamentan en canales encubiertos de almacenaje ya que éstos permiten almacenar mayor cantidad de información en un flujo que codificarla en la diferencia de tiempo entre los paquetes del mismo. Una de las principales desventajas de aquellos canales basados en el tiempo consiste en que necesitan gran cantidad de paquetes para poder enviar la información, lo que les hace más susceptibles a la detección. Nuestro sistema de comunicación, al que denominamos *Invisible Covert Channel* (InCC) [91], puede ser clasificado como un canal de almacenamiento que se adapta dinámicamente al tráfico de la red.

2.6. Modelos de mitigación

Este apartado tiene por objeto aportar una visión general de la mitigación de ataques DDoS. Para ello, en primer lugar, presentamos una clasificación de las soluciones de mitigación en la que se agrupan funcionalidades (ver tabla 2.3), a continuación discutiremos las diversas soluciones y técnicas que aportan los autores y, finalmente, clasificaremos las diferentes técnicas de mitigación (tabla 2.4).

En la tabla adjunta a continuación se aporta una clasificación de los sistemas que se basan en soluciones en Router y aquellas otras que no. Se ha realizado esta distinción en base al tipo de elemento físico principal, que en el caso de las redes sería el Router. Las primeras se dividen en las tres categorías que hemos considerado más relevantes, si emplean soluciones distribuidas, con gestión de colas o información de protocolo, y el resto de soluciones las hemos dividido en aquellas basadas en puzzles, con filtrado y distribuibuidas.

		Autores con sistemas de mitigación
Basado en Router	Distribuido	Ioannidis [38], Chen [92], Fariba [93], DLimiter [94]
	Colas Protocolo	Zhang [95], Jen [96], Ioannidis [38], Jarmo [97], Fariba [93], DLimiter [94] Badishi [98], Safva [99], Lu [100], Supranamaya [101]
No basado en Router	Distribuido	Mankins [34], Shi [102]
	Puzzles	Wang [103], Fraser [104]
	Filtrado	Goldstein [105]

Tabla 2.3: Clasificación de sistemas de mitigación

A continuación, se detallan cada uno de los métodos propuestos por los autores ya mencionados.

Goldstein *et al.* [105] proponen un método de mitigación basado en la teoría de redes bayesianas. Su sistema no detecta el DDoS pero genera reglas de filtrado a nivel IP para Firewalls como iptables [106], nf-HiPAC o listas de acceso de sistemas Cisco y con dichas reglas los servidores pueden continuar dando servicio a los usuarios legítimos durante un ataque.

Safva *et al.* [99] sugieren una defensa en los Routers borde que permite determinar si la dirección IP origen está falseada o no y, así, poder determinar si el segmento TCP SYN-ACK es válido. Para ello emplean una tabla de estados que relaciona los segmentos TCP SYN y los SYN-ACKS, de forma que si un segmento SYN-ACK no es válido entonces el Router borde reinicia la conexión TCP. De esta forma la cola de peticiones del servidor, también denominada Backlog Queue, se libera y el servidor puede atender a clientes legítimos.

El estudio de Mankins *et al.* [34] propone una arquitectura de Routers distribuida y un protocolo de pago, de forma que cobra a la red de origen y al servidor los costes de iniciar una conexión con el servidor.

Lu *et al.* [100] proponen obtener las características de la conexión extremo a extremo en una red, como latencia, jitter, etc... Mediante la aplicación de dichas métricas sobre conexiones normales se obtienen las características de éstas y, mediante transformadas de Fourier, el algoritmo empleado reconstruye las características de las conexiones y aporta una estimación de lo que sería una conexión normal a una anormal.

Badishi *et al.* [98] muestran como, mediante el empleo del *Security Parameter Index* (SPI) aleatorios, es posible incrementar la resistencia del protocolo IPsec durante un ataque y, de esta forma, reducir el número de operaciones criptográficas que penalizan el rendimiento de la conexión.

Supranamaya *et al.* [101] consideran los ataques DDoS que cumplen con los estándares de los protocolos no intrusivos y que utilizan aplicaciones legítimas para saturar los recursos de un sistema. Proponen un planificador que está integrado en una arquitectura Proxy, que intercepta las peticiones maliciosas antes de que lleguen a los servidores de forma que éstos no se saturen.

Shi *et al.* [102] proponen un sistema denominado OverDoSe, que consiste una red virtual que ofrece protección frente a ataques. OverDoSe ofrece aislamiento a nivel IP a los Routers que forman parte de la red, de forma que regulan el tráfico al servidor mediante los mensajes de éste. Este sistema puede ser desplegado en un ISP que requiera protección frente a ataques y necesite ofrecer servicio a sus clientes de red. Como podemos ver en la figura 2.13, OverDoSe

básicamente consiste en una red IP transparente en la que, para poder establecer una conexión con el servidor web, primero es necesario resolver un puzzle y, una vez resuelto, el sistema encamina el tráfico de dicho usuario a través de la red mediante el uso de Cookies en las cabeceras HTTP.

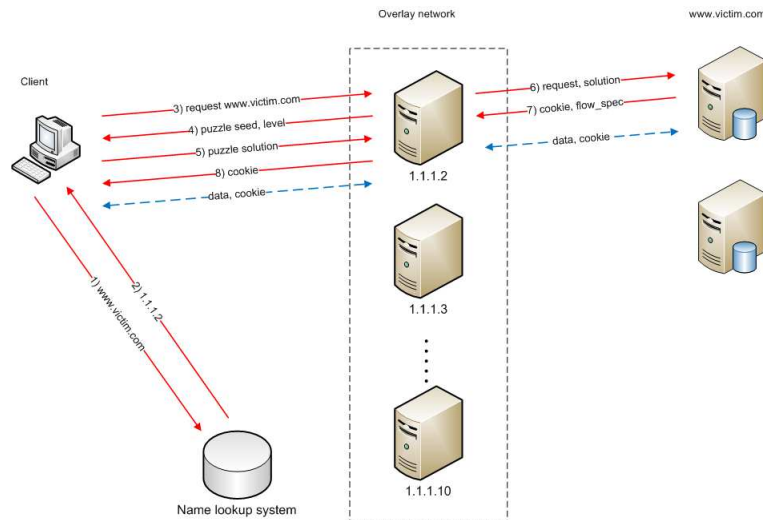


Figura 2.13: Protocolo básico de OverDoSe

Fariba *et al.* [93] muestran un sistema basado en el empleo de colas, denominado *Integrity Based Queueing* (IBQ). Su modelo incorpora firmas hash a cada paquete por cada Router por el que pasa, de modo que cada uno de ellos gestiona sus colas en base a dichas hashes. El problema de este modelo reside en el elevado coste de CPU y de latencia de los flujos debido al cálculo de las funciones hash.

Otras soluciones a considerar son las basadas en puzzles, también conocidas como Captchas, que se implementan en los servidores objetivo.

Por ejemplo, Wang *et al.* [103] sugieren la utilización de puzzles de congestión (*Congestion Puzzles*, CP), una nueva contramedida para ataques que consumen en ancho de banda disponible. Tal y como ocurre con las defensas basadas en puzzles, CP fuerza a los atacantes a consumir grandes recursos de los atacantes para que éstos no puedan llevar a cabo el ataque. El CP está diseñado para ataques de consumo de ancho de banda, más frecuentes en la capa IP, sin embargo en el núcleo de la red resulta más elegante utilizar puzzles distribuidos para que los Routers puedan cooperar y verificarlos.

El modelo que proponen Fraser *et al.* [104], también basado en puzzles, se denomina *Memoryless Puzzle Protocol* (MPP). El objetivo de MPP es ofrecer una solución viable que permita mitigar el DDoS en entornos de routing anónimos como puede ser la red TOR [107]. Su investigación se fundamenta en la exploración de ataques DDoS basados en *Transport Layer Security* (TLS) sobre

dicha red.

Jarmo *et al.* [97] analizan la efectividad del rate-limiting como mecanismo de mitigación cuando hay un DDoS, y estudian el efecto en las conexiones legítimas que tienen que ser preservadas. Demuestran que el rate-limiting es un buen mecanismo para mitigar el DDoS sin penalizar el tráfico legítimo.

Jen *et al.* [96] sostienen que la separación de redes constituye una mejor aproximación para un sistema de encaminamiento que la eliminación de redes de dicho sistema global.

Basado en la catalogación de tráfico, también denominado *Traffic Shaping*, tenemos otro trabajo de Goldstein *et al.* [108] en el que catalogan cientos de direcciones IP origen y aplican diferentes anchos de banda mediante el empleo de un algoritmo similar a *Random Early Detection* (RED), con el que se consiguen resultados muy parecidos al RED.

Zhang *et al.* [95] proponen un sistema capaz de generar una red más robusta y resistente basada en un mayor aporte de inteligencia a los Routers que lo integran. Esta red se puede desplegar en Internet y puede ser usada para detectar, detener y recuperarse de un ataque. De hecho, permite detectar un ataque antes de que éste ralentice los flujos dirigidos al objetivo. Estos Routers pueden descartar paquetes IP selectivamente, especialmente los más próximos al atacante, de forma que los servidores atacados puedan continuar ofreciendo sus servicios.

Chen *et al.* [92] presentan dos mecanismos de defensa perimetrales para los *Internet Service Providers* (ISPs) que no requieren soporte de los Routers externos o internos del ISP. Con el objeto de bloquear el ataque, dichos mecanismos permiten la cooperación de los Routers borde, identificando los flujos de los atacantes y aplicando rate-limiting a los mismos. Así mismo, los autores también estudian un nuevo mecanismo de IP traceback [109] y demuestran analíticamente, mediante simulaciones, que su solución reacciona rápidamente al bloque de tráfico y que presenta un elevado grado de supervivencia para los flujos legítimos.

A modo de conclusión, en la tabla 2.4 se clasifican los autores citados en base a la funcionalidad que proponen en sus soluciones, de manera que 'Drop' cuando los paquetes IP son eliminados, 'Rate-limiting' cuando la solución utiliza algún tipo de gestión de colas para controlar el flujo, 'Statistics' cuando se emplea algún tipo de estadística, 'ACL' cuando utilizan listas de control de acceso, 'Micropaying' cuando se cobra algún tipo de coste a los usuarios o servidores a modo de micro pago por la utilización de la red y Puzzles si identifican que la conexión realizada contra un servidor ha sido efectuada por un usuario legítimo.

Autor	Router	Distribuido	Colas	Protocolo	Descarte	Limitar velocidad	Estadísticas	ACLs	Micro-pagos
Goldstein [105]	*						*	*	
Safva [99]	*			*					
Mankins [34]		*							*
Lu [100]	*		*	*			*		
Bandishi [98]	*			*					
Supranamaya [101]	*		*	*					
Fariba [93]	*	*	*						
Shi [102]		*		*					
Jarmo [97]	*					*			
Jen [96]	*		*			*			
Ioannidis [38]	*	*	*						
Zhang [95]	*				*				
Cheng [92]	*	*							
DLimiter [94]	*	*	*		*	*			

Tabla 2.4: Técnicas de mitigación

Nuestro sistema DLimiter [94], encargado de mitigar el ataque, presenta las siguientes características: está basado en Router, puede funcionar de manera distribuida en varios Routers o cores, incluye una gestión de colas independientes, limita los flujos y descarta paquetes IP selectivamente.

2.7. Necesidades y carencias actuales

Prácticamente todos los meses se reciben noticias de ataques llevados a cabo sobre cualquier tipo de infraestructura. Al mismo tiempo cabe resaltar que prácticamente ninguno de los sistemas empleados en la actualidad es seguro frente a dichos ataques ni dispone de capacidad como para resistirlos. Se citan, a continuación, varios de los ataques efectuados durante el último cuatrimestre del año 2012, así como otros más recientes materializados durante del 2013.

Durante octubre del 2012 las tensiones entre Japón y China aumentaron tras la decisión del gobierno de Noda al comprar y nacionalizar las Islas Senkaku. Como consecuencia de ello, durante dos semanas, varias infraestructuras japonesas, gubernamentales y locales, junto con bancos, universidades y empresas fueron objeto de diversos ataques DDoS [110] lanzados desde China.

Otro ejemplo a mencionar lo constituye el ataque DDoS que lanzaron los ciberactivistas contra las infraestructuras del banco HSBC como consecuencia de la crisis económica global y que provocó la interrupción de los servicios de dicha entidad financiera.

Así mismo, otras infraestructuras, como el portal de intercambio de ficheros "The Pirate Bay" también ha sido objeto de un ataque DDos [111] recientemente.

Otro caso lo constituyen las infraestructuras americanas, como el "Bank of America" [112], que fue atacado por un grupo musulmán denominado 'Innocence of Muslims'.

Así mismo, el 20 de marzo del 2013, el proveedor de servicios en la nube CloudFlare [113] sufrió un ataque de reflexión DNS [114] que llegó a superar los 300Gbps. Como puede observarse en la figura 2.14, a día de hoy y según

los datos de los que se dispone, éste es el ataque más grande de la historia de Internet.

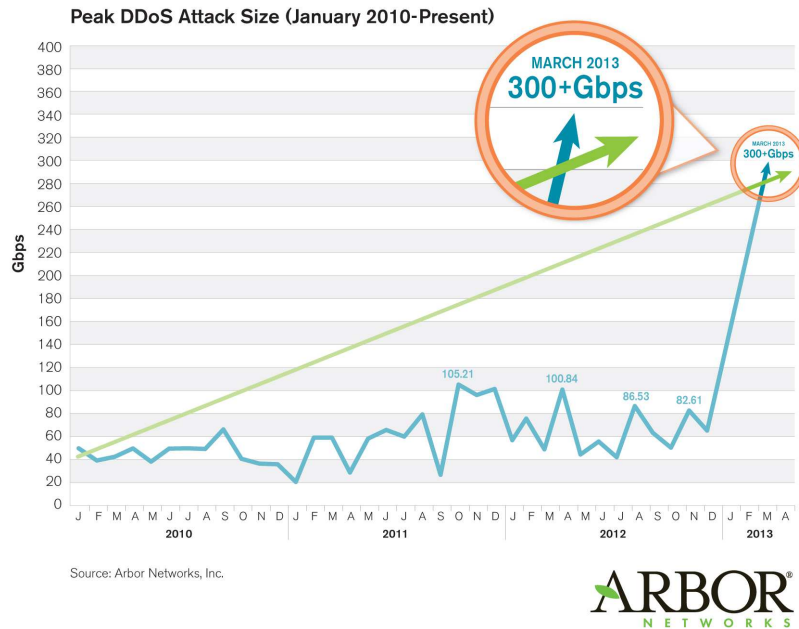


Figura 2.14: Histórico de ataques DDoS [13]

Los citados ataques se producen en cualquier parte del mundo, existiendo siempre detrás de ellos una clara componente económica [115] y reivindicativa. No obstante, además de las agresiones mencionadas, la mayoría de los últimos ataques que se producen son de naturaleza militar y en ellos el objetivo es diferente. Generalmente los autores de este tipo de agresiones suelen ser grupos organizados y distribuidos. En este sentido cabe mencionar que, si bien organizaciones como Anonymous, KDMS Team o Luzsec suelen hacerse responsables de los mismos, las organizaciones gubernamentales nunca lo hacen, aún cuando existan grandes evidencias de su autoría.

Capítulo 3

Arquitectura propuesta

En este capítulo vamos a detallar la arquitectura de la solución propuesta, así como los diferentes elementos que la componen. Uno de los pilares sobre los que se sustenta la arquitectura consiste en la necesidad de que ésta sea distribuida, de manera que sea escalable, desplegable y permita proteger tanto sistemas básicos como grandes infraestructuras ISPs.

La propuesta que se presenta se compone de tres subsistemas que cooperan entre sí:

1. El subsistema de mitigación, encargado de controlar el flujo de los usuarios a los servidores finales.
2. El subsistema de detección, responsable de la detección de los posibles ataques a la infraestructura de servidores.
3. El subsistema de comunicación, cuya finalidad consiste en comunicar los dos subsistemas anteriores con otro tipo de sistemas, de manera que dichas comunicaciones no puedan ser interceptadas ni alteradas.

Estos sistemas pueden encontrarse en cualquier nodo de la red, ya que son totalmente independientes, y pueden distribuirse de la forma que se requiera. Obviamente, dado que son sistemas seguros, éstos no deberían ser accesibles a nivel IP por ningún elemento de la red, salvo que éstos se encuentren aislados por una red física de gestión.

A continuación, en el apartado 3.1 presentamos un esquema de la solución propuesta que incluye una descripción de los servicios que se ofrecen, seguidamente en el apartado 3.2 proponemos la integración del sistema en la red de la Universidad Rey Juan Carlos, detallándose las diferentes VLANs y, finalmente, en el apartado 3.3 se exponen las conclusiones sobre la arquitectura propuesta.

3.1. Esquema de la solución

La solución que presentamos se compone de tres sistemas claramente diferenciados (mitigación, detección y comunicación), que pueden ser integrados o desplegados en diferentes nodos de la red.

La figura 3.1 muestra la solución propuesta en la que se disponen tres nodos, en cada uno de los cuales se despliegan dos procesos. Con la finalidad de que los diferentes nodos puedan cooperar entre sí, éstos se comunican mediante un proceso que permite ocultar la información a compartir entre ellos y, al mismo tiempo, evita que puedan ser atacados ya que carecen de direccionamiento IP.

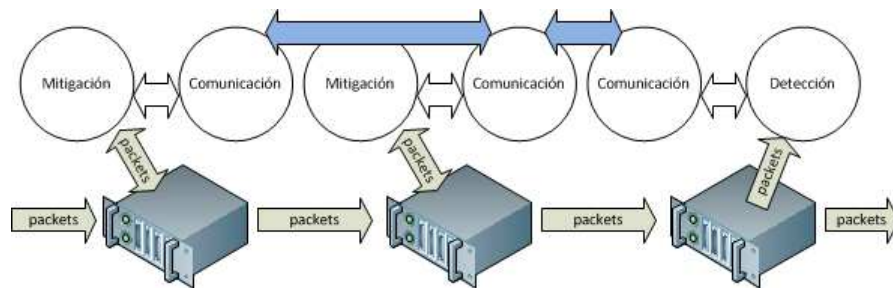


Figura 3.1: Procesos

El principal objetivo del proceso de mitigación consiste en detener el impacto del ataque DDoS reduciendo, en la mayor medida posible, las pérdidas en la calidad del servicio a los usuarios legítimos. El proceso de detección es el encargado de decir quién está realizando un ataque y quién no. Y, finalmente, el proceso de comunicación aporta una capa de comunicación que garantiza que, aunque la red de producción se vea comprometida por un usuario avanzado, éste no pueda acceder al sistema bajo ningún concepto.

Se ha empleado el sistema de intercomunicación D-Bus [116] ya que su utilización permite integrar los procesos mediante el sistema de comunicación más estándar empleado en Linux y, al mismo tiempo, permite realizar llamadas síncronas y asíncronas, enviar señales y monitorizar los procesos de manera sencilla e independiente.

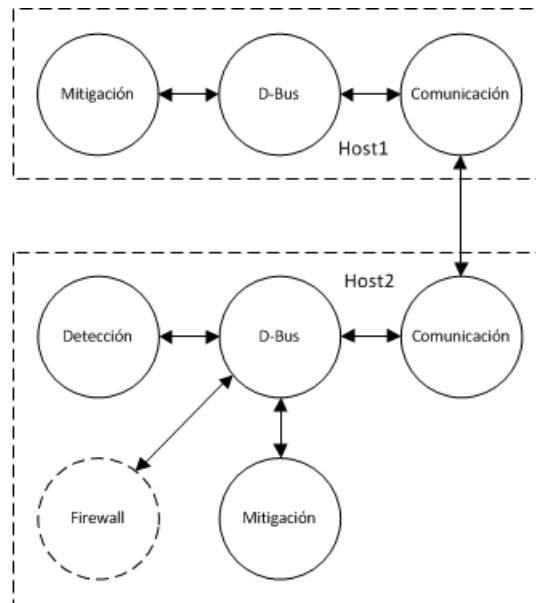


Figura 3.2: Procesos cooperando

En la figura 3.2 mostramos dos modelos de distribución de procesos. El Host1 incluye un proceso de mitigación y otro de comunicación, procesos que se comunican entre sí por medio de los servicios que éstos exportan. Por otra parte, el Host2 se compone de los procesos de detección, mitigación y comunicación, así como otro proceso externo denominado Firewall. El Host1 y el Host2 cooperan entre sí de manera que, cuando el proceso de detección del Host2 identifica un ataque DDoS, éste lo notifica por medio de una señal a su sistema de comunicación para que lo transmita a su proceso de mitigación y a su Firewall, así como al Host1. El número de procesos de cada Host depende del número de CPUs disponibles en el nodo y del nivel de carga de la red. En la figura anterior puede apreciarse, a modo de ejemplo simplificado, la distribución de un modelo de procesos, no obstante, éste es extensible a tantos procesos como CPUs tenga el nodo.

Servicios Con la finalidad de que los procesos anteriores puedan comunicarse entre sí, éstos ofrecen una serie de servicios que pueden ser invocados por otros procesos o, incluso manualmente, por el propio administrador de la red.

A continuación, en el código 3.1, se describen los servicios del sistema de comunicación.

```

static ST_Callback ST_StaticInccEngineMethods [] = {
    {
        .name = "Start",
        .in = NULL,
        .out = "b",
        .func = PRCA_Method_StartEngine
    },
    {
        .name = "Stop",
        .in = NULL,
        .out = "b",
        .func = PRCA_Method_StopEngine
    },
    {
        .name = "SetSourceDevice",
        .in = "s",
        .out = "b",
        .func = PRCA_Method_SetSourceDevice
    },
    {
        .name = "ShowAvailableChannels",
        .in = NULL,
        .out = "a(si)",
        .func = PRCA_Method_ShowAvailableChannels
    },
    {}
};

```

Código C 3.1: Servicios de comunicación

Los servicios exportados del sistema de comunicación más relevantes son: *Start* para iniciar el proceso y para que éste se mantenga en funcionamiento, *Stop* para detener la ejecución del proceso, *SetSourceDevice* para indicar qué dispositivo de red tiene que analizar el tráfico y, finalmente, *ShowAvailableChannels* para mostrar los canales encubiertos disponibles en un instante determinado.

Con estos servicios, cuando el sistema de mitigación quiera reportar un incidente, éste enviará una señal denominada *ReportIncident* al D-bus, de manera que cualquier proceso con un manejador para dichas señales (ver código 3.2) podrá adoptar diferentes acciones en función de si procesa la señal o no.

```

static ST_Callback ST_StaticSignals [] = {
    {
        .name = "ReportIncident",
        .in = "s",
        .out = NULL,
        .func = SignalHandler_ReportIncident
    },
    {}
};

```

Código C 3.2: Manejador de incidentes

Los servicios que exporta el sistema de mitigación (ver código 3.3) serán utilizados por el sistema de detección para cambiar a los usuarios de colas o para incrementar la carga de CPU de los nodos de mitigación.

```

static ST_Callback ST_StaticDlimiterEngineMethods[] = {
    {
        .name = "SetQueueDelay",
        .in = "i",
        .out = "b",
        .func = PRCA_Method_SetQueueDelay
    },
    {
        .name = "CreateQueue",
        .in = NULL,
        .out = "b",
        .func = PRCA_Method_CreateQueue
    },
    {
        .name = "DestroyQueue",
        .in = "i",
        .out = "b",
        .func = PRCA_Method_DestroyQueue
    },
    {
        .name = "CreateThreadOnQueue",
        .in = "i",
        .out = "b",
        .func = PRCA_Method_CreateThreadOnQueue
    },
    {
        .name = "DestroyThreadOnQueue",
        .in = "i",
        .out = "b",
        .func = PRCA_Method_DestroyThreadOnQueue
    },
    {
        .name = "GetUserQueue",
        .in = "s",
        .out = "i",
        .func = PRCA_Method_GetUserQueue
    },
    {
        .name = "SetUserQueue",
        .in = "si",
        .out = "b",
        .func = PRCA_Method_SetUserQueue
    },
    {}
};

```

Código C 3.3: Servicios de mitigación

Los servicios más relevantes del sistema de mitigación son: *SetQueueDelay* que modifica el tiempo de retraso de una cola del sistema, *CreateQueue* que crea una cola en el sistema en caso de necesidad dependiendo de la carga existente, *DestroyQueue* que elimina una cola del sistema, *CreateThreadOnQueue* que genera un hilo de ejecución en una cola especificada como parámetro, *DestroyThreadOnQueue* que permite suprimir un hilo de ejecución en una cola específica, *GetUserQueue* que devuelve el identificador de cola de un usuario

dado vía dirección IP origen y, finalmente, *SetUserQueue* que cambia de cola a un usuario con el objeto de penalizarlo o gratificarlo.

A continuación, en las figuras 3.3 y 3.4, mostramos un diagrama de secuencia con un incidente detectado en el Host2 que envía información al Host1.

Cuando el proceso de detección identifica un ataque DoS, ya sea distribuido o no, éste envía una señal al D-bus denominada *ReportIncident*. En nuestro caso, disponemos de tres procesos suscritos a dicha señal, un Firewall externo, el proceso de comunicación y el de mitigación. El D-bus reenviará la señal *ReportIncident* a estos procesos para que tomen las acciones adecuadas y el proceso de comunicación reenviará dicha señal al Host1 mediante el canal encubierto que encuentre disponible.

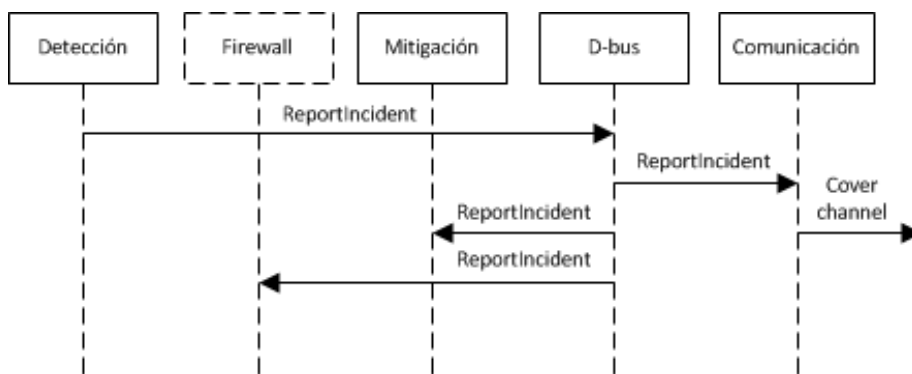


Figura 3.3: Diagrama de secuencia Host2

El Host1 recibe, a través del proceso de comunicación, la señal *ReportIncident* del Host2 y, en este caso, ejecuta el método *SetUserQueue* para penalizar o gratificar a un usuario, tal y como se muestra en la figura 3.4, o a más usuarios en cuyo caso existirían más llamadas a dicho método.

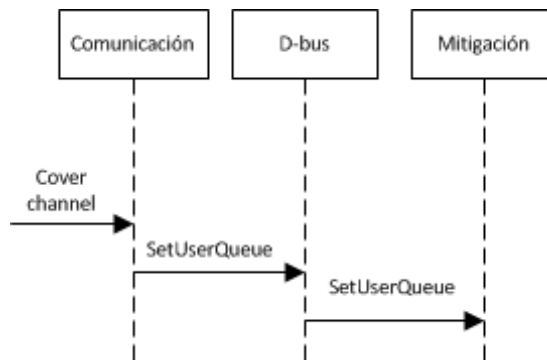


Figura 3.4: Diagrama de secuencia Host1

3.2. Integración en la red real

Con el objeto de poder desplegar completamente la arquitectura que se propone hemos integrado en la red de la universidad los diferentes elementos necesarios para detectar, mitigar y comunicar un ataque de denegación de servicio, ver figura 3.5.

La topología de la red de la universidad dispone de los siguientes elementos:

- Un Firewall con funciones de encaminamiento y de protección a la red de la universidad.
- Un Router que, junto con otros Routers, encamina el tráfico a diferentes campus de las universidades de Madrid.
- Un Switch GigaEthernet conectado al Router anterior.
- Switchs pertenecientes a cada edificio del campus que, por medio de VLANs [117] estáticas, se conectan a las diferentes subredes a nivel IP, de forma que en una VLAN tenemos los PCs de alumnos, en otra los servidores web externos, en otra los servidores SMTP y en otra las bases de datos, entre otros.

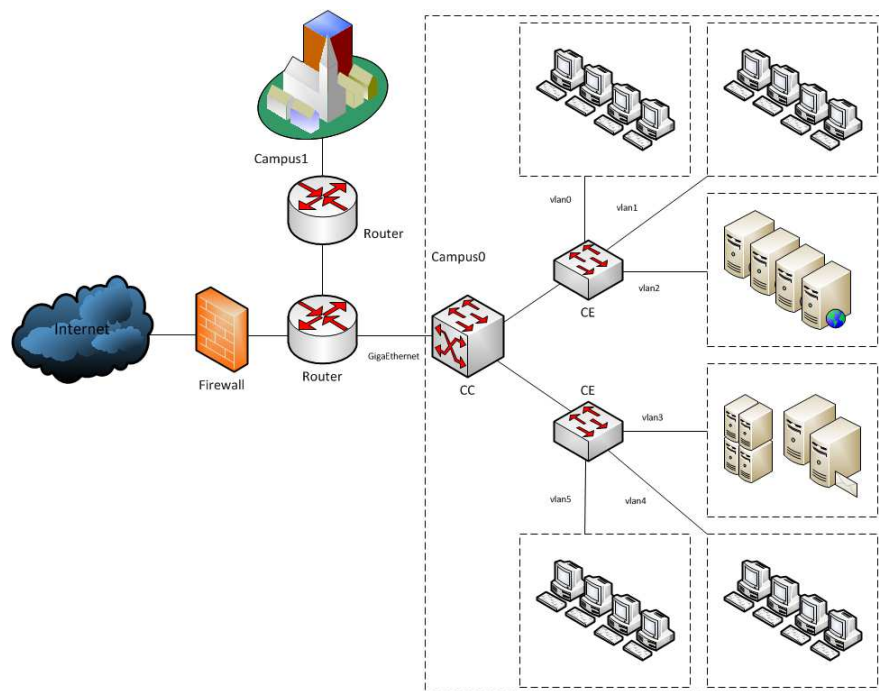


Figura 3.5: Red de la Universidad Rey Juan Carlos

Mediante esta topología de red, desplegamos los Routers donde se encuentran los diferentes sistemas propuestos. Concreamente, teniendo en cuenta que lo que se quiere proteger son los servidores web de la vlan2 y vlan3 , proponemos la inclusión de dos sistemas de detección, tres sistemas de mitigación y cuatro sistemas de comunicación (ver figura 3.6).

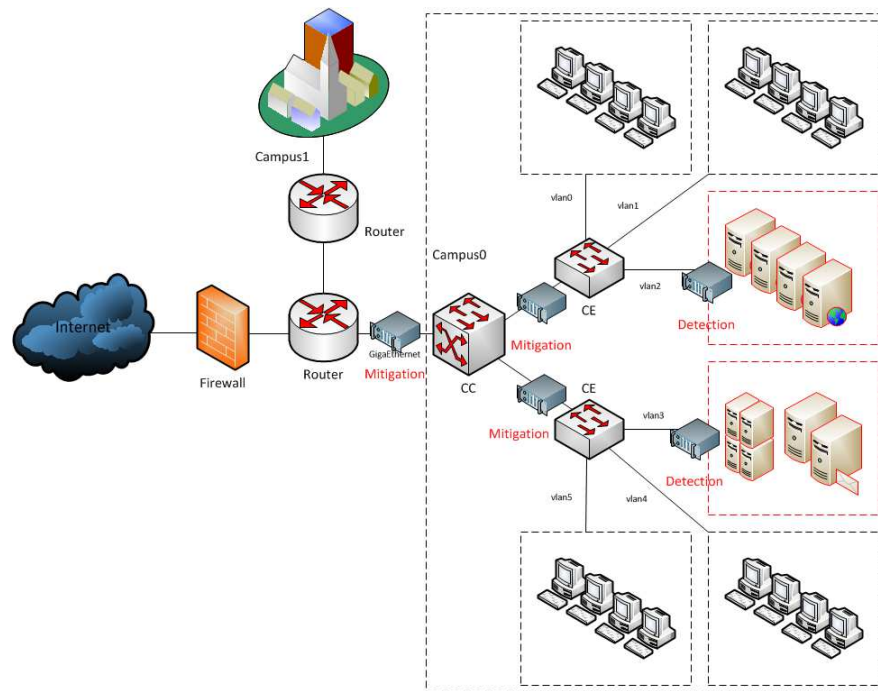


Figura 3.6: Red de la Universidad Rey Juan Carlos con modelo propuesto

A continuación se explica el despliegue completo en una de las VLANs, concretamente en la vlan2 (ver figura 3.6), ya que en la vlan3 éste se realizaría de idéntica forma.

Como se muestra en la figura, el despliegue del sistema de detección es el encargado de notificar las direcciones IP que muestran un comportamiento no adecuado o ilegal sobre los servidores web de la vlan2. Dichas notificaciones son enviadas a través del sistema de comunicación, responsable a su vez del envío de la información correspondiente a los diferentes sistemas de mitigación situados entre el Router de acceso y el Switch CC, así como entre los Switchs CC y CE respectivamente. De esta manera, la solución propuesta considera dos sistemas de mitigación dispuestos en cascada que permiten el reparto de cargas entre los sistemas de mitigación facilitando, de esta manera, un despliegue mucho más robusto en caso de producirse un ataque DDoS.

3.3. Conclusiones

En este capítulo hemos propuesto una arquitectura distribuida en la que los diferentes nodos que la integran, junto con los procesos, pueden estar situados en cualquier lugar de la red.

Concretamente, los procesos que la componen pueden ejecutarse concurrentemente, dependiendo del número de CPUs de cada nodo. Así mismo, según la infraestructura de la red, cada uno de los procesos puede tener una funcionalidad distribuida, de manera que los procesos de detección puedan estar situados en los mismos servidores web a proteger o en routers intermedios, en función de las necesidades específicas de cada red.

Cabe destacar que la arquitectura con servicios a la que se hace referencia en el subapartado 3.1 dota al sistema de gran versatilidad, lo que nos permite integrarla en otros servicios externos, como Routers, Firewalls, NIDS, etc.. de una manera rápida y eficaz.

Por otra parte, con la finalidad de analizar grandes cantidades de tráfico, es necesario que el sistema sea escalable de manera que, dependiendo de la topología de la red y de los servicios a proteger, éste pueda adaptarse a las necesidades de la red, especialmente en lo relativo a los términos económicos.

Capítulo 4

Detección de ataques DDoS

4.1. Planteamiento del problema

Descripción del problema

Uno de los principales problemas de la detección de denegación de servicio radica en la diferenciación de los flujos que proceden de un usuario legítimo de aquellos generados por uno ilegítimo. Para ello, abordamos el problema desde la perspectiva de un ataque DDoS en el que tendremos que detectar un gran flujo de datos hacia un servidor.

Problemas existentes

Una de las principales dificultades en la detección de ataques DDoS consiste en la gestión de los falsos positivos y de los falsos negativos. Un falso positivo se refiere a la detección de un usuario como atacante por parte del sistema cuando en realidad no lo es y tiene el inconveniente de que el usuario legítimo se verá penalizado y será considerado como ilegítimo. Por otra parte, consideramos como falso negativo a la detección de un usuario legítimo cuando en realidad no lo es, lo que pone de manifiesto la existencia de un atacante que no ha sido detectado.

Solución propuesta

Se propone abordar el problema de la diferenciación del tráfico legítimo del que no lo es, distinguiendo entre aquel generado por un humano y que constituye una navegación normal del susceptible de constituir un ataque, como por ejemplo un ataque de una Botnet, WebCrowds, etc...

En el apartado 4.2 del presente capítulo describimos la solución propuesta, así como las diferentes partes que lo componen. Seguidamente, en el apartado 4.3 mostramos las pruebas realizadas y los resultados obtenidos y, finalmente, en el apartado 4.4 se mencionan las conclusiones obtenidas a partir del sistema propuesto.

4.2. Solución

En base a la arquitectura distribuida que se propone se opta por el empleo de un demonio del sistema, que será el encargado de realizar dicha detección.

La solución que se aporta funciona como un sensor que analiza los flujos que se dirigen a los servidores HTTP [8] y, para ello, pueden disponerse varios sensores que trabajan independientemente y que reportan las incidencias a un sistema de mitigación o incluso a un Firewall.

El principal objetivo de la solución propuesta consiste en detectar un ataque DDoS a un servidor HTTP y para llevar a cabo dicha tarea se realizan tres análisis diferentes: análisis estadístico, análisis de grafo HTTP y análisis de camino HTTP. Como se detalla más adelante, cada uno de ellos detecta un subtipo de ataque y todos ellos se aplican a cada uno de los flujos HTTP que van al servidor. Cuando uno de estos análisis identifica un comportamiento fuera de lo normal, se etiqueta la IP origen como sospechosa con el objeto de notificarlo a un sistema de mitigación, como puede ser un Firewall. De esta manera, si uno de los análisis detecta un comportamiento anómalo los otros análisis ya no se llevarán a cabo.

El primer análisis a realizar es el estadístico cuyos valores forman parte de la capa TCP/IP y también de la capa HTTP, de modo que la combinación de dichos valores nos ayuda a discriminar y conocer el comportamiento. Los estadísticos que se consideran para la detección son número de paquetes por flujo, flujos por usuario, peticiones por usuario, etc... Los referidos valores estadísticos se comparan mediante media y desviación típica con unos valores iniciales y, si éstos sobrepasan los valores iniciales, la dirección IP origen se marca como sospechosa.

El segundo análisis, que denominamos cache de grafo HTTP (CGH), se lleva a cabo siempre que se realice el primer análisis y consiste en el procesamiento de las peticiones que lleva el flujo HTTP. Estas van generando un grafo con las diferentes peticiones GETS y POST del flujo, donde los nodos son la URL y la diferencia de tiempo entre una petición y otra es el coste, representado por el arco del grafo. Los flujos que se guardan en la HTTP cache son flujos legítimos que actualizan la cache, de manera que cuando llega un flujo HTTP ilegítimo, que no tiene por qué ser un ataque, éste debería seguir uno de los caminos del grafo y debería hacerlo con una duración mayor a la existente en el grafo. En caso de que el flujo siga un camino sin arcos en la cache y lo realice con una duración inferior a la existente en el grafo, dicha IP se marcará como sospechosa.

El tercer análisis que se realiza lo denominamos análisis de caminos frecuentes HTTP y se almacena en la llamada cache de caminos HTTP (CCH). Este análisis comprueba cuantas veces se van repitiendo los caminos, de forma que si un camino se repite demasiado entonces se etiqueta como sospechoso.

El diseño del algoritmo propuesto es el resultado de la combinación de los tres análisis anteriores.

En él se efectúa el análisis estadístico en primer lugar ya que, en relación con los otros dos análisis, éste tiene un consumo despreciable de memoria y de CPU. Así mismo, al detectarse la mayoría de los ataques distribuidos en esta fase

del análisis estadístico, su realización en primer lugar aporta robustez y solidez al algoritmo. Por otra parte, el segundo y tercer análisis, que corresponden al análisis de la cache de grafo y al análisis de caminos, dependen entre sí ya que sin grafo no existirían caminos.

Cada uno de estos análisis detecta un subtipo de ataque que se detallará más adelante de forma que combinando todos estos análisis llegamos a la conclusión del algoritmo aquí expuesto.

A continuación, en la figura 4.1, se muestran las diferentes partes del algoritmo y cómo se estudiaría el flujo HTTP en función de los resultados obtenidos.

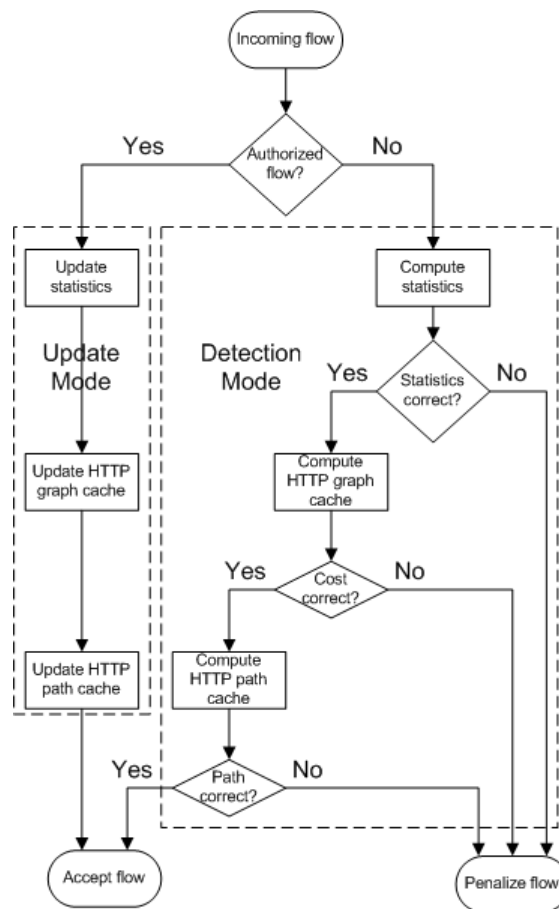


Figura 4.1: Algoritmo de detección [66]

Las decisiones del algoritmo anterior, *Statistics correct?*, *Cost correct?* y *Path correct?*, pueden ser parametrizadas de manera que aporten más flexibilidad al algoritmo permitiendo, al mismo tiempo, ajustarlo de manera más precisa durante la detección. Estos valores de parametrización, se actualizan mediante

el *Update mode*, de manera que éstos se auto ajusten automáticamente y el algoritmo pueda ir cambiando su comportamiento de manera dinámica.

La principal función de *Update mode* consiste en disponer de los parámetros anteriormente mencionados lo más actualizados y realistas posible. Para ello, y con la finalidad de que dichos valores sean reales, utilizamos lo que denominamos flujos autorizados. Estos flujos se caracterizan por disponer de una dirección IP origen que forma parte de una lista de acceso de manera que las comunicaciones de dicha IP podrán indirectamente actualizar dichos valores sin introducir ningún tipo de latencia en dichos flujos. Para autorizar los flujos es necesario que la dirección IP origen sea previamente autenticada en el servidor web destino, ya sea mediante SSL [118], Captchas [103], usuarios frecuentes del servicios, administradores u otros mecanismos existentes de autenticación disponibles en la actualidad.

4.2.1. Análisis estadístico

Considerando que el empleo de estadísticos complejos provocaría una degradación del sistema en términos de consumo de CPU se han seleccionado los siguientes estadísticos por su simplicidad de computación a la hora de calcularlos:

- Número de peticiones GET.
- Media de peticiones GET.
- Desviación típica de peticiones GET.
- Número de peticiones POST.
- Media de peticiones POST.
- Desviación típica de peticiones POST.
- Media de flujos por usuario.
- Desviación estándar de flujos por usuario.
- Media de flujos por minuto y por usuario.
- Media de peticiones por minuto y por usuario.

Con la finalidad de modelar el comportamiento de nuestro servidor, es posible emplear un mayor número de valores estadísticos siempre que dichos valores estén en consonancia con la capacidad de los servidores a proteger y con la complejidad de cálculo de los mismos.

La combinación de estadísticos de la capa TCP/IP y HTTP aporta robustez al modelo propuesto. En este sentido, cabe destacar que un elevado número de expertos en el tema [53, 60, 63, 65] considera únicamente el uso de una de estas capas. La correlación entre un gran número de flujos creados accediendo a la misma URL por un usuario nos lleva a pensar que dichos estadísticos tiene que combinarse entre ellos, aportando así mayor robustez a la detección.

Mediante el empleo de dichas estadísticas es posible determinar el comportamiento del servidor web y descartar un gran número de flujos y de usuarios que se comportan de manera anómala en comparación con su comportamiento habitual. Dichas estadísticas pueden actualizarse mediante listas de acceso, de forma que usuarios que accedan de forma habitual a los servidores, estén autenticados, y utilicen correctamente los recursos, puedan actualizar las estadísticas globales. De esta manera, aunque el contenido del servidor web sea dinámico, los estadísticos pueden variar al mismo tiempo.

4.2.2. Análisis de la cache de grafo HTTP (CGH)

La cache de grafo HTTP (CGH) es una zona de memoria en la que residen las hashes de las URLs del servidor web, como se puede observar en el código 4.1. Dichas hashes pueden ser aprovisionadas mediante un fichero pcap [119] o bien mediante un proceso interactivo de usuarios autorizados. Un fichero pcap es un archivo que contiene una imagen exacta de los paquetes IP que pasaron por uno o varios dispositivos de red, que permite reproducir con exactitud lo que sucedió en la red en un instante determinado.

En el código 4.1 se muestra la estructura de la CGH que procedemos a explicar. La *ST_GraphCache* contiene una tabla hash denominada *uris*, que contiene a su vez *ST_GraphLinks*, que incluyen otra tabla hash también denominada *uris* que contiene la estructura final *ST_GraphNode*.

```
enum node_types {
    // A simple resource as a image.
    NODE_TYPE_REGULAR = 0,
    // The resource implies cpu has access to database, cgi, etc...
    NODE_TYPE_MEDIUM,
    // The resource is a big file that implies
    // bandwidth consumption, also a POST with the upload
    NODE_TYPE_BIG
} ;

struct ST_GraphNode {
    gchar *uri; // stores the HTTP uri
    int key;
    int cost;
    int32_t hits;
    enum node_types type;
} __attribute__((packed));

typedef struct ST_GraphNode ST_GraphNode;

struct ST_GraphLink {
    GHashTable *uris; // contains ST_GraphNode structs
    gchar *uri;
    int key;
    int hited;
    enum node_types type;
} __attribute__((packed));

typedef struct ST_GraphLink ST_GraphLink;
```

```

struct ST_GraphCache {
    GHashTable *uris; // contains ST_GraphLink structs
    int32_t total_links;
    int32_t total_hits;
    int32_t total_fails;
    int32_t total_nodes;
    int32_t total_node_hits;
    int32_t total_node_fails;
    int32_t total_ids;
    int statistics_level;
    int32_t size_memory; // total bytes allocated
} __attribute__((packed));

typedef struct ST_GraphCache ST_GraphCache;

```

Código C 4.1: Estructuras de la CGH

En la CGH, ver figuras 4.2 y 4.3, se almacenan las hashes de las diferentes URLs y los diferentes tiempos entre cada petición HTTP y puede tener los siguientes modos de funcionamiento, el modo actualización para actualizar los valores y el modo consulta donde se comprueban los valores del flujo con los de la CGH.

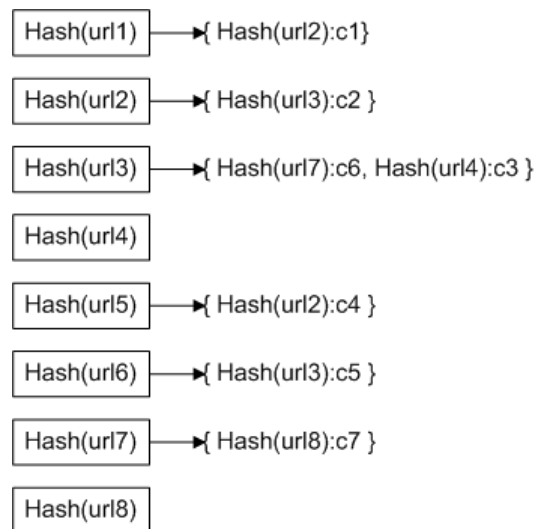


Figura 4.2: CGH en memoria

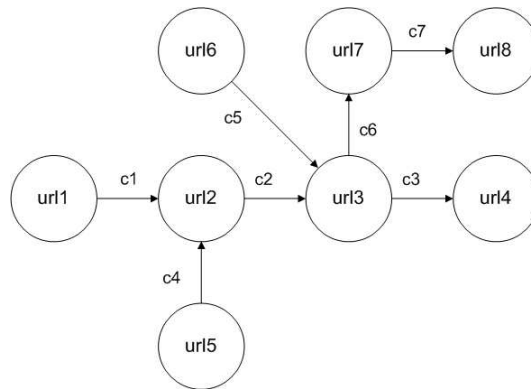


Figura 4.3: Grafo de navegación sobre una página web

Modo actualización En este modo los flujos autorizados vía lista de acceso serán los que actualizarán la CGH de manera que, una vez que un flujo haya sido autorizado, se examinará cada cabecera HTTP del mismo. Para ello, se calculan los diferentes tiempos de las diversas peticiones, así como las hashes de las URLs, almacenándose dichos valores en la CGH en forma de grafo (ver figura 4.3 y 4.4).

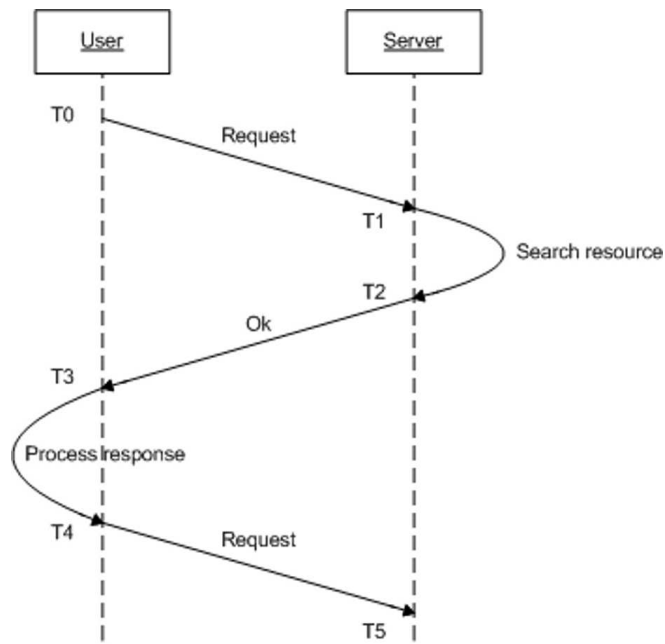


Figura 4.4: Petición y respuesta HTTP [66]

De esta forma, cuando el proceso de análisis del flujo finaliza, se obtiene un grafo dirigido con las hashes y los intervalos de tiempo entre peticiones. Como se muestra en la figura 4.4, la diferencia de tiempos indica el coste del cliente que procesa la página web enviada por el servidor y la diferencia entre T5 y T1 representa, en este caso, el coste completo de la totalidad del flujo. De esta forma, al almacenar las diferencias de tiempo entre peticiones es posible detectar cuando un usuario está teniendo una diferencia de tiempo anormal, identificándolo así como usuario de un ciberataque.

Así mismo, la solución propuesta considera un caso especial en el que el navegador descarga un recurso pesado, como por ejemplo un fichero de gran tamaño. Para esta operación el funcionamiento habitual de muchos navegadores consistiría en la apertura de varios flujos en un período de tiempo reducido para intentar descargarse el fichero, generándose falsos positivos. Como se puede apreciar en la figura 4.5 un elevado pico de nuevos flujos se dan en el instante 1000 para la descarga un fichero. Es precisamente en ese instante donde se generaría el falso positivo, pero nuestro sistema es capaz de almacenar dicha información en la CGH de forma que para ese fichero se pueda incrementar el límite de flujos por usuario para dicho recurso.

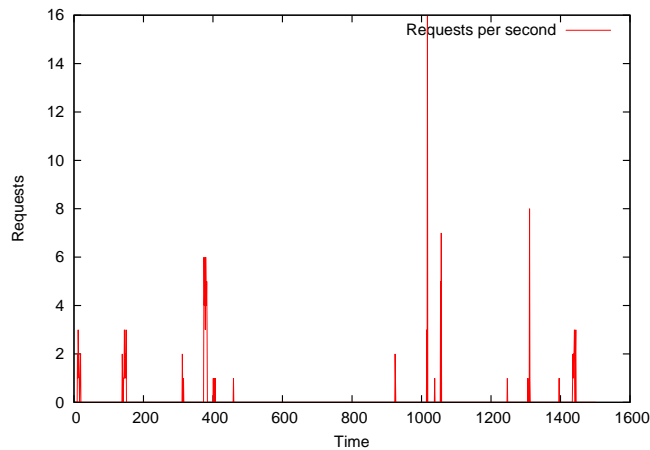


Figura 4.5: Peticiones de usuario de un recurso pesado

4.2.3. Análisis de la cache de caminos HTTP (CCH)

Para aquellos ataques basados en la repetición de caminos y en temporizadores, en los que el patrón de acceso se repite constantemente, hemos desarrollado un detector de patrones de acceso en los flujos que permite identificar este problema. Una de las herramientas que utiliza esta técnica para lanzar ataques basados en temporizadores es Slowloris. Prácticamente casi todas las herramientas DoS utilizan repetición de alguna forma, así como Siege que también tiene dicha funcionalidad.

Tal y como se muestra en el código 4.2, la CGH corresponde a una estructura en la que el camino que siguen las peticiones HTTP del flujo se almacenan en un *ST_PathNode*. De esta manera, por medio de un acceso hash, podemos saber si dicho camino ya existía o si, en su caso, está siendo repetido.

```

struct ST_PathNode {
    gchar *path;
    int32_t hits;
} __attribute__((packed));

typedef struct ST_PathNode ST_PathNode;

struct ST_PathCache {
    GHashTable *paths; // contains ST_PathNode structs
    int32_t total_paths;
    int32_t total_hits;
    int32_t total_fails;
    int statistics_level;
    int64_t size_memory; // total bytes allocated
} __attribute__((packed));

typedef struct ST_PathCache ST_PathCache;

```

Código C 4.2: Estructuras de la CCH

A continuación, mostramos tres figuras en las que se puede observar un ataque y, al mismo tiempo, lo comparamos con una actividad normal en un servidor web. Dichas figuras contienen respectivamente en los ejes *X* e *Y* el tiempo y el número de caminos diferentes que siguen las peticiones HTTP recibidas en un segundo, aportando una mejor visión del modo en que se produce un ataque DDoS.

Como se muestra en la figura 4.6, el número de caminos repetidos por segundo es muy elevado en comparación con el número de caminos repetidos en el servidor (ver figura 4.8). Un efecto similar puede observarse en la figura 4.7 con la herramienta Slowloris e incluso con Loic. Así mismo, la figura 4.6 muestra que el número de caminos repetidos es muy elevado en comparación con el de la figura 4.8 en la que, si bien existe un mayor número de usuarios, éstos discurren por caminos de navegación diferentes. Ambas figuras muestran que, tanto en Loic como en Slowloris, la manera de acceder a los diversos caminos es diferente como resultado de su diferente implementación.

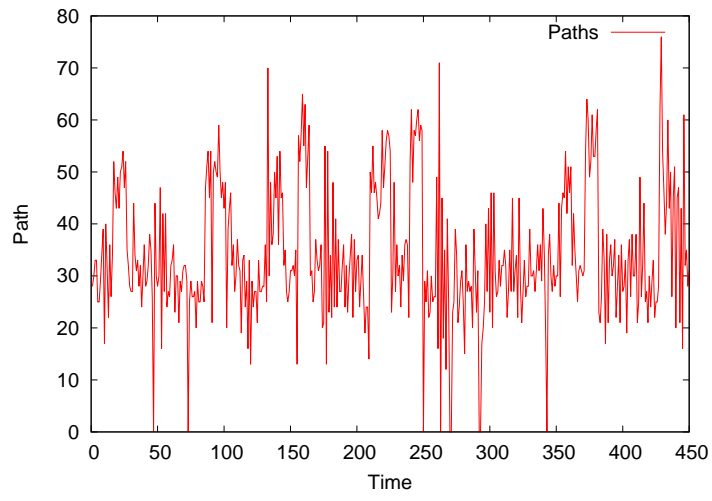


Figura 4.6: Frecuencia de caminos HTTP en Siege [66]

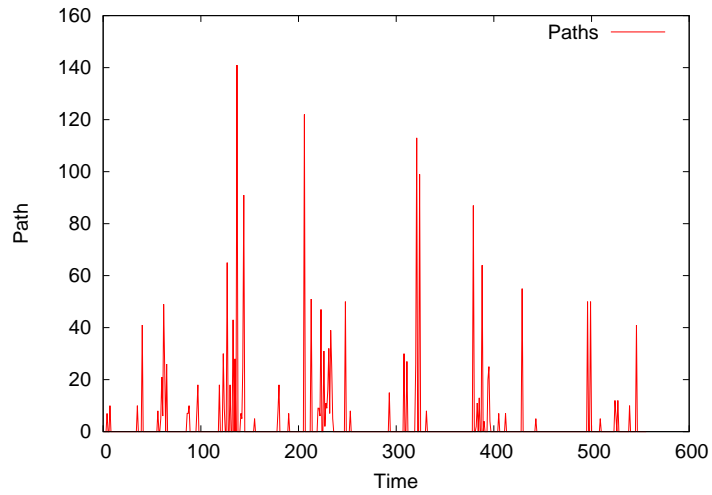


Figura 4.7: Frecuencia de caminos HTTP en Slowloris [66]

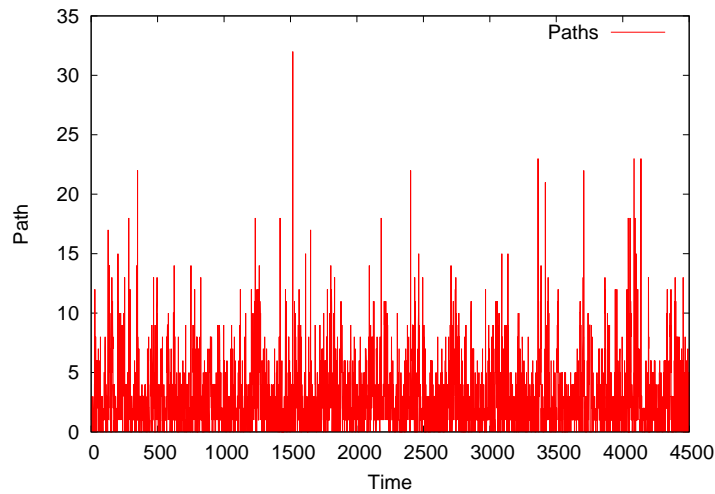


Figura 4.8: Frecuencia de caminos HTTP en un servidor web [66]

En la siguiente figura 4.9 se muestra el proceso completo de las dos caches anteriores, así como un ejemplo del modo en que se correlacionarían los flujos con dicha estructura. En la parte izquierda de ésta disponemos de un diagrama de grafo de una navegación web a un servidor que se estructura en la *HTTP graph cache* (CGH) y, seguidamente, en la *HTTP path cache* (CCH). De forma que podemos tener todos los diagramas de navegación entre diferentes URLs que un usuario ha llevado a cabo. Es importante resaltar que la comprobación de las URLs en la cache, así como de los diferentes costes y caminos, se realiza en $O(1)$, lo que nos permite tener un rendimiento de CPU muy elevado ya que la complejidad es constante.

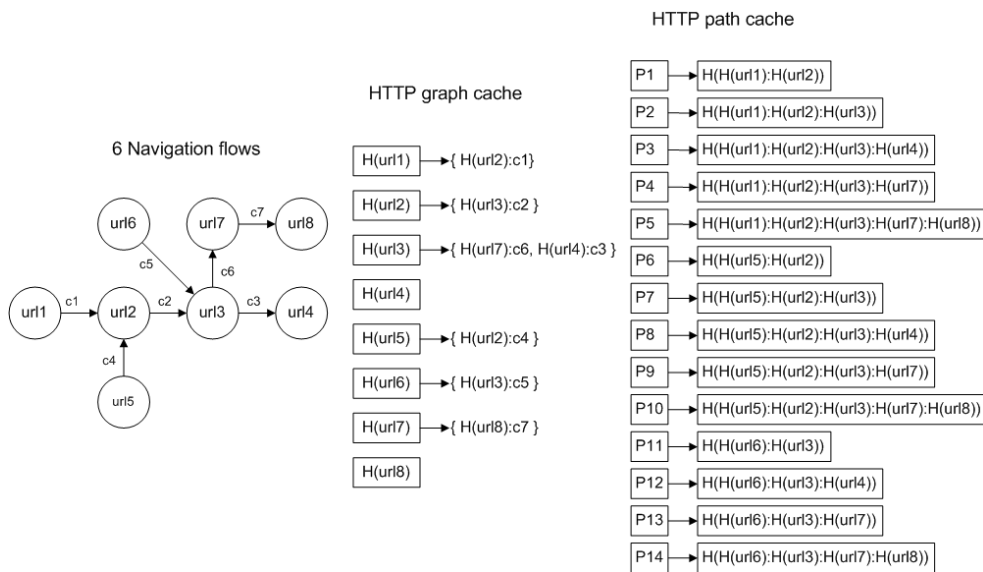


Figura 4.9: Caches HTTP

Con las técnicas anteriormente descritas el sistema es capaz de detectar los siguientes casos a nivel de aplicación:

- Caminos aleatorios (`http://<target>/random_path`). Esta técnica permite acceder mediante patrones aleatorios a las diferentes URLs de la página web. Como resultado de ello, y como consecuencia de los fallos de cache existentes por no estar almacenadas las páginas que se solicitan, se genera un elevado consumo en la CPU del servidor.
- Acceso de entrada (`http://<target>/login.php`). Este caso tiene como principal objetivo generar una elevada tasa de acceso a la base de datos, basándose en la necesidad del usuario de validarse en el sistema.
- Búsqueda de información (`http://<target>/search.php`). El principal objetivo de este ataque consiste en provocar una elevada frecuencia en los scripts que utilice el servidor para la búsqueda de información en las bases de datos.
- Operaciones POST. Estas operaciones generalmente afectan a la memoria RAM ya que, cuando un usuario sube un fichero, éste tiene que almacenarse temporalmente en memoria. Las operaciones POST están relacionadas con lo que se denominan ataques de pulso [120, 121, 122], que consisten en utilizar el mecanismo de retransmisión TCP para ralentizar los flujos de subida al servidor y disminuir la memoria disponible en el mismo.

4.3. Experimentos y resultados

Para la evaluación del sistema propuesto se han empleado trazas de un servidor real, con diferentes dominios alojados (servidor multidominio). Estas trazas han sido capturadas en diferentes días del mismo mes. El servidor web es un servidor Apache con soporte PHP y para los test hemos procesado 290.000 flujos de 14.000 usuarios diferentes. Los datos reales empleados nos han permitido evaluar el sistema en situaciones reales, con un elevado número de usuarios accediendo a un servidor web público de Internet.

Para ello, en primer lugar y con la finalidad de generar el modelo, hemos procesado las trazas del servidor, de forma que las cache HTTP y las estadísticas están cargadas en memoria. A continuación, hemos inyectado al modelo diferentes trazas de Siege, Slowloris y Loic con los diversos tipos de ataques que éstas ofrecen y, finalmente, hemos comprobado la detección del algoritmo, ver figura 4.1, para ver su efectividad. El desarrollo del algoritmo ha sido llevado a cabo partiendo de la implementación de Polyvaccine [123], donde se ha desarrollado un módulo de detección DDoS compatible con la funcionalidad de este. Polyvaccine es un sistema que permite detectar exploits polimórficos desconocidos en servidores web. Desarrollando el algoritmo propuesto sobre la base de Polyvaccine ampliamos la funcionalidad, permitiéndole detectar ataques DDoS sobre servidores web.

4.3.1. Modelo de tráfico del servidor web cacheado

El tráfico empleado para la realización de los experimentos corresponde al de un servidor web Español de dominio público que tiene funcionando un Apache con soporte PHP.

Las trazas fueron tomadas durante diferentes días de la semana, concretamente dos de ellas en días laborables y la tercera en un día festivo, con la finalidad de disponer de mayor actividad (ver figuras 4.10, 4.11 y 4.12).

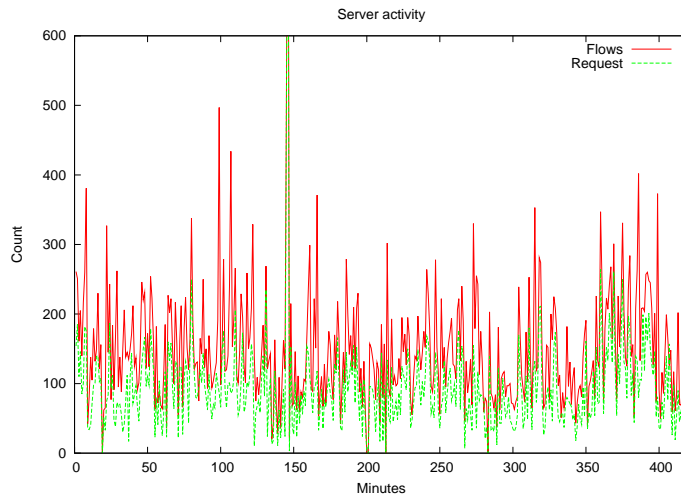


Figura 4.10: Actividad del servidor en el primer día

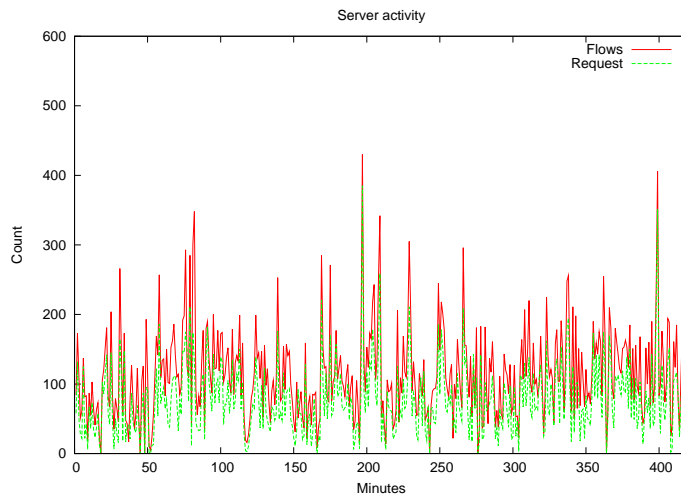


Figura 4.11: Actividad del servidor en el segundo día

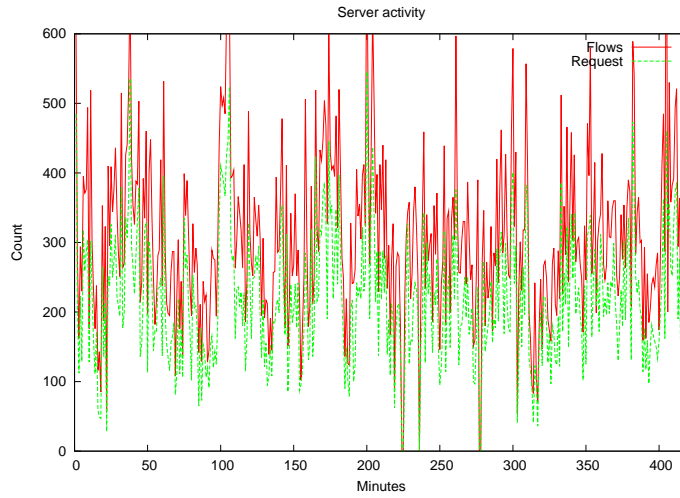


Figura 4.12: Actividad del servidor en el fin de semana

Uno de los valores más interesantes para demostrar experimentalmente la viabilidad de la propuesta, que se muestra en la tabla 4.1, corresponde al consumo CPU requerido por el sistema de detección para el procesamiento de los flujos, y que refleja un consumo prácticamente despreciable.

Muestra	Duración	Usuarios	Flujos	Peticiones	Memoria RAM	Consumo CPU
Día normal	7 horas	2317	42807	63582	332 Mbytes	20 segs
Día normal	7 horas	1676	33253	52172	332 Mbytes	21 segs
Fin de semana	24 horas	10939	217410	288728	339 Mbytes	1,3 min

Tabla 4.1: Actividad del servidor web y consumo del algoritmo de detección

En primer lugar se han analizado las estadísticas, cuyos valores pueden verse en las tablas 4.2 y 4.3, tomándose como referencia la media y la desviación típica de los estadísticos mencionados en el apartado 4.2.1.

Media flujos	Desviación flujos	Media GETs	Desviación GETs	Media POSTs	Desviación POSTs
9	52	30	75	1	0

Tabla 4.2: Estadísticas globales

Media flujos/min	Desviación flujos/min	Media request/min	Desviación request/min
7,45	24,9	1,52	5,26

Tabla 4.3: Estadísticas de tiempo globales

Con la finalidad de verificar la eficiencia de las estructuras de cache anteriores (CGH y CCH), detalladas en los apartados 4.2.2 y 4.2.3, hemos calculado su efectividad basándonos en tráfico real y en los aciertos y fallos que ésta genera. Como observamos en la figura 4.13, si aumentamos el número de fallos de cache, también denominada tolerancia a fallos, aumenta el porcentaje de efectividad de la misma. Esto significa que el sistema tolera que no existan algunas peticiones HTTP en la cache sin ser, por tanto, tan estricto a la hora de la detección.

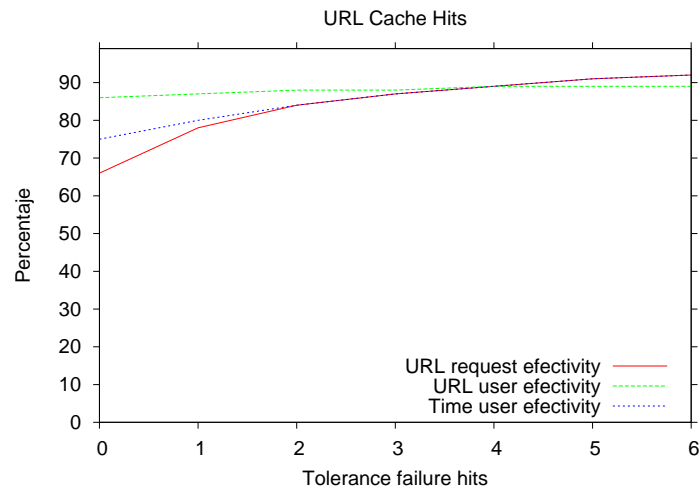


Figura 4.13: Efectividad de la cache de URLs [66]

La figura 4.13 muestra cómo un incremento de la tolerancia a fallos incrementa la efectividad de ésta, que puede llegar a alcanzar el 90 %, y una diferencia de las trazas de 24 horas, de lo que se desprende que al poder cachearse prácticamente el 90 % de los flujos, la diferencia de comportamiento de un día respecto al otro es prácticamente igual, salvo los fines de semana en el que el servidor presenta mayor actividad.

Sin embargo, para obtener tasas mayores y valores cercanos al 99 % de efectividad, pueden emplearse Bots o autorizaciones dinámicas de usuario, que actualizan los valores de la cache sin que esto suponga coste adicional para el resto de flujos. La "sensibilidad" de la cache también es tolerante a los fallos de cache, lo que nos permite gestionar los falsos positivos que ésta pueda ocasionar, ya que si una URL no ha sido accedida y no figura en la cache y un usuario realiza un acceso a dicha URL, entonces generará un fallo en la misma que provocaría un falso positivo.

4.3.2. Comportamiento del modelo frente ataques

En este apartado se utilizan trazas generadas en laboratorio de Siege, Slowloris y Loic, que se inyectarán en el sistema con el objetivo de identificar las partes

del algoritmo en las que éstas son detectadas. Como puede observarse en la tabla 4.4, al menos cuatro de los casos generados son identificados mediante el empleo de estadísticas, lo que demuestra que la elección de los estadísticos resulta crucial para una buena detección. En este sentido, cabe destacar que el empleo de estas herramientas con algo más de conocimiento permitiría saltarse las estadísticas y, de esta manera, probar otras partes del algoritmo.

Sin embargo, como muestran los resultados de la tabla 4.4, en determinadas situaciones el análisis estadístico no es suficiente. Por ello, la utilización de otras opciones que incluyen las herramientas nos permite ir ejecutando las diversas partes del algoritmo y dar cobertura a todas las opciones que éstas ofrecen.

El sistema propuesto contiene un módulo geográfico que determina a que país corresponde una dirección IP, de manera que si nuestro sistema estuviera completamente saturado por un ataque a gran escala podríamos utilizar esta técnica para descartar flujos. No obstante, este módulo no ha sido verificado ya que con el algoritmo de detección cubrimos las expectativas. En cualquier caso, la viabilidad de dicho módulo sería relativamente fácil ya que, como puede verse en la figura 4.14, la distribución de tráfico que recibe el servidor web resulta bastante homogénea por parte de los usuarios nacionales, lo que permite que en caso de un ataque desde otro país éste pudiera ser fácilmente detectado.

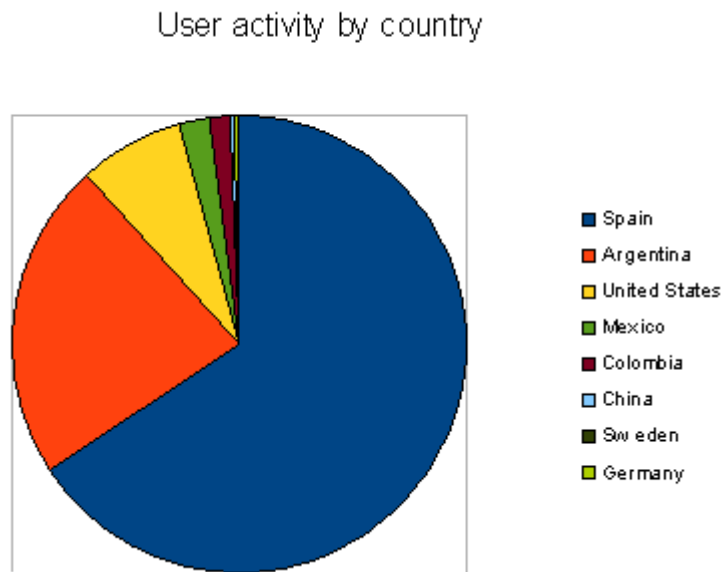


Figura 4.14: Distribución por países de los usuarios

La mayoría de las herramientas empleadas en la investigación permiten variar la frecuencia del ataque, así como otras funciones. No obstante, se han intentado cubrir las opciones más interesantes para poder así tener una visión más géner-

ica. Por ejemplo, el caso Siege lento con repetición de camino genera múltiples flujos con caminos diferentes que se van reproduciendo en el mismo flujo HTTP.

Herramienta	Estadísticos	CGH	CPH
Slowloris	detectado		
Siege	detectado		
Siege lento con pagina aleatoria	no detectado	detectado	
Siege lento con repetición de pagina	no detectado	detectado	
Siege lento con repetición de camino	no detectado	no detectado	detectado
Loic modo rápido	detectado		
Loic modo lento	detectado		

Tabla 4.4: Herramientas versus algoritmo

4.4. Conclusiones

El sistema de detección que se propone combina la utilización de diferentes estadísticos de diversas capas de red, el uso de grafos para detectar si las peticiones del flujo se encuentran en la cache y el análisis de caminos frecuentes para evitar su repetición por parte de los usuarios. Cabe destacar que todas las técnicas anteriormente expuestas permiten detectar gran parte de los ataques actuales. Concretamente, el modelo propuesto también ha descubierto actividad de Bots, como la del bot de Google [124], y así como una auditoría ilegal por medio de la herramienta Acunetix [125] que se utiliza para auditar webs.

Así mismo, cabe resaltar que el sistema puede ser implementado de manera que, con la finalidad de evitar saturaciones de CPU, la funcionalidad de detección pueda distribuirse por los diferentes nodos de la red.

Otro aspecto importante a resaltar reside en la independencia del sistema propuesto con respecto al contenido del servidor web, facilitando el acoplamiento a otros servidores web. Para finalizar, conviene mencionar que la integración con otros sistemas de seguridad resulta sencilla ya que está acoplado al sistema de alertas que proporciona Linux.

Capítulo 5

Comunicaciones seguras

5.1. Planteamiento del problema

Descripción del problema

El principal problema en la comunicación a través de redes TCP/IP de dos sistemas seguros tales como Firewalls, Proxies o NIDS, entre otros, radica en que, a pesar de que éstos envían información sensible, como por ejemplo SSL [118], éstos pueden ser detectados en una red. Como consecuencia de ello, un atacante puede identificar los servicios que ofrecen y atacarlos, además de deshabilitar las funciones de protección de las redes y apropiarse de información (mediante inyección de código SQL), instalando rootkits, expandir virus, etc...

Problemas existentes

El principal problema consiste en que todos los sistemas de comunicación dejan rastro. De manera que, mediante un seguimiento del mismo resulta relativamente sencillo descubrir las direcciones IP involucradas en la comunicación. De forma que, una vez conocidas las direcciones IP, se podrían lanzar ataques DDoS a las mismas con el fin de ralentizar e interrumpir las comunicaciones.

Solución propuesta

Se propone una solución denominada *Invisible Covert Channel* (InCC) que permite la comunicación entre dos o más sistemas mediante el empleo de canales encubiertos de almacenamiento. Dichos sistemas deben enviar y recibir información confidencial de manera segura a través de redes no confiables. Para ello, se combinan diferentes técnicas que permiten que los sistemas que quieren enviarse información entre sí lo realicen de manera segura sobre redes TCP/IP, de forma totalmente transparente, sin comprometer dichos sistemas e impidiendo que puedan ser descubiertos bajo ningún tipo de análisis de red, como podría ser mediante el empleo de un Sniffer [119].

En el apartado 5.2 del presente capítulo exponemos una descripción de la solución, a continuación en el apartado 5.3 mostramos los resultados obtenidos y, finalmente, en el apartado 5.4 presentamos las conclusiones del sistema propuesto.

5.2. Solución

Como podemos observar en la figura 5.1, la idea principal de la solución que se expone consiste en generar paquetes IP, idénticos a los ya mostrados, que se camuflan en otro tráfico existente en la red, como puede ser tráfico bittorrent. InCC utilizará las direcciones IP origen y destino de *A* y de *B* de manera que el canal encubierto no tendrá direccionamiento IP y, como consecuencia de ello, tanto el *Sender* como el *Receiver* serán totalmente invisibles en la red.

El hecho de que los elementos *Sender* y *Receiver* no tengan dirección IP en una red significa que éstos no pueden ser accedidos por ninguna máquina vía protocolo IP. De esta forma, aunque éstos capturen el tráfico específico necesario para mimetizarse, también pueden generar datagramas IP pero con direcciones de otros elementos de la red, como detallaremos más adelante.

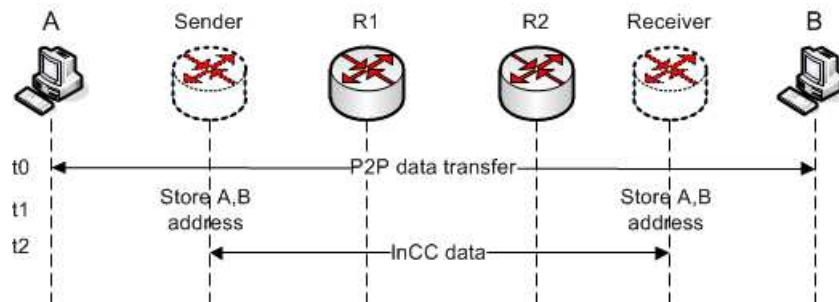


Figura 5.1: Mecanismo de selección de dirección IP [91]

En este apartado se procede con la descripción de la solución InCC, (apartado 5.2.1), a continuación, detallaremos las diferentes partes del protocolo (apartados 5.2.2, 5.2.3, 5.2.4 y 5.2.5). Finalmente vamos a presentar un prototipo de implementación (apartado 5.2.6).

5.2.1. Descripción del protocolo

Con el objeto de impedir la detección, InCC soporta las siguientes especificaciones:

- Utiliza la técnica que denominamos 'Port-walking' inspirada en port-knocking [126, 127, 128], que consiste en el empleo de puertos de manera aleatoria para impedir cualquier tipo de análisis. Las aplicaciones P2P y VoIP utilizan frecuentemente este tipo de técnicas.

- Cifrado de la parte principal del mensaje mediante RC4 [129], con la introducción de algunas variaciones como rotación de clave. Se opta por RC4 debido su simplificación de código y escaso consumo de CPU.
- Las direcciones IP origen y destino son falseadas, de modo que únicamente origen y destino las conocen.
- El empleo de firmas de tráfico de otras aplicaciones, tales como Snort [130] y OpenDPI [131], son utilizadas por el sistema para camuflarse en otro tipo de tráfico de red.

La figura 5.2 muestra el funcionamiento del protocolo. En ella origen y destino pueden encontrarse en dos redes diferentes o en la misma y en la que A y B representan a dos usuarios que están realizando una transferencia de fichero mediante una aplicación P2P (ver figura 5.1) y origen y destino están autenticados mediante RSA [118] o port-knocking. Si bien nuestra implementación utiliza autenticación estándar RSA, no obstante esta fase no está indicada en la figura ya que suponemos que ambos usuarios están autenticados.

Una vez autenticados tanto origen como destino, éstos se encuentran preparados para recibir mensajes en los puertos X e Y , siendo los elementos A y B usuarios de P2P. Se describen a continuación cada uno de los pasos propuestos:

1. El origen detecta una sesión P2P entre A y B cuya identificación se lleva a cabo mediante firmas.
2. El origen envía un mensaje al destino incluyendo la siguiente información:
 - Identificador P2P que reconoce la firma detectada en la sesión P2P.
 - Dirección IP origen (`src_ip`) y puerto origen (`src_port`) de A .
 - Dirección IP destino (`dst_ip`) y puerto destino (`dst_port`) de B .
 - Posible acuse de recibo.
 - La totalidad del mensaje va cifrado mediante RC4.
3. Una vez que el mensaje es recibido por el destino, y los mensajes son decodificados, éste modifica su comportamiento.
4. El destino envía un mensaje de acuse de recibo porque el origen activó dicho flag. Este mensaje de respuesta se compone de bytes aleatorios generados por el destino que dificultan la realización de un análisis del tráfico de la red.
5. El origen recibe el acuse de recibo y entiende que los nuevos mensajes que se generen deben ir con la dirección y puertos de A y B , así como con la firma con la que se detectó la sesión P2P.
6. El origen envía un datagrama con firma P2P y con la carga útil cifrada con RC4, siendo A y B las direcciones origen y destino.

7. El receptor intercepta el mensaje, lo descifra, decodifica y lo envía a una capa superior para informar de ello, en este caso del incidente. Finalmente, el receptor no encamina y elimina dicho mensaje, ya que el usuario *B* podría generar datagramas *ICMP unreachable port* que podrían ser sospechosos y delatar al sistema.

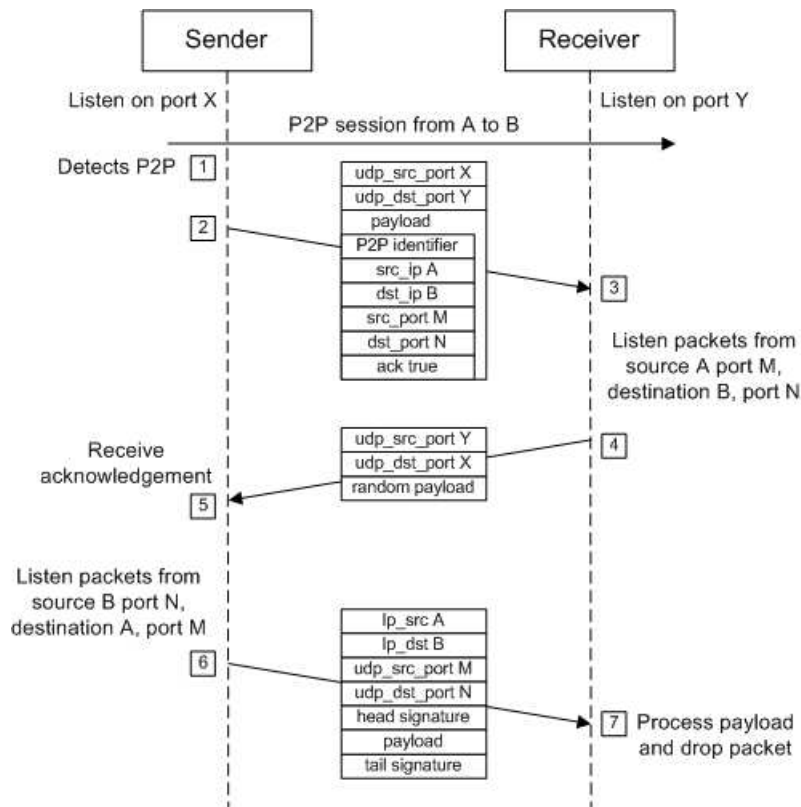


Figura 5.2: Protocolo InCC [132]

Tras la descripción del protocolo, se procede con la definición de las partes que componen el sistema. Estas son: 'Port Walking' cuando el flujo de datos va cambiando por diferentes puertos, 'Payload noise' que incorpora ruido a la carga útil del paquete, 'IP randomness' que genera direcciones IP aleatorias o utiliza las vistas en la red y, finalmente, 'Signature poisoning' que añade firmas de tráfico conocido en el paquete generado.

5.2.2. Port walking

Port knocking [126, 127, 128] es una técnica utilizada para autenticar equipos en Internet que consiste en la combinación de paquetes a puertos cerrados o

abiertos de tipo SYN de modo que, si la combinación es correcta, se habilita un servicio de SSH que permite el acceso al Host. Es decir, se habilitan o deshabilitan servicios ocultos, de manera que éstos únicamente son autorizados mediante la combinación de paquetes especiales. Un elevado número de sistemas seguros emplean este tipo de autenticación que permite acceder a sistemas remotos de forma segura.

De este modo, mediante el mismo principio de *Port knocking*, se ha creado lo que denominamos Port walking (Andar por puertos) (ver figura 5.3). Esta técnica se basa en la generación de un flujo que va cambiando de puertos, dando lugar a un mayor número de flujos. Mediante el empleo de este principio, InCC se comporta como una aplicación P2P en la que se genera un elevado número de flujos para que las técnicas de filtrado y de gestión de ancho de banda no resulten útiles.

Por otra parte, a cada objeto que se envía al destino, se le añade un campo indicando el siguiente puerto. Si el objeto a reportar ocupa un elevado número de bytes, InCC lo dividirá en varios fragmentos. Esta funcionalidad es especialmente recomendada cuando BitTorrent [133, 134], Gnutella, Skype y aplicaciones similares son detectadas, ya que éstas utilizan la selección de puertos aleatorios para la señalización y transferencia de ficheros, entre otros.

Como puede verse en la figura 5.3, para sincronizar el destino con los puertos que se van a seleccionar, InCC envía los siguientes puertos (*src_port* para el origen y *dst_port* para el destino), de forma que el destino cambiará su puerto al valor *Z* y el origen únicamente escuchará a través del puerto *W*. El valor de los puertos puede ser seleccionado de manera aleatoria o reutilizando los puertos detectados en una sesión P2P.

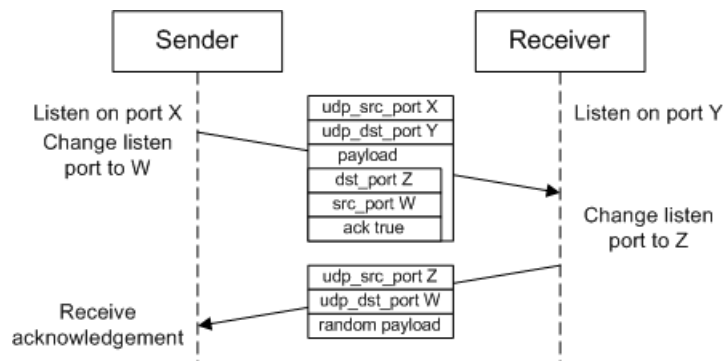


Figura 5.3: InCC port walking [91]

En este caso la salida del comando `tcpdump` [119] muestra la técnica sin acuse de recibo en la que 192.168.1.1 representa el origen y 192.168.1.2 el destino, teniendo como puerto de inicialización el 2000.

```
IP 192.168.1.1. 47578 > 192.168.1.2. 2000: UDP, length 123
IP 192.168.1.1. 35690 > 192.168.1.2. 7030: UDP, length 157
```

A continuación se muestra la salida de tcpdump con acuse de recibo. En este ejemplo el acuse utiliza el mismo puerto y los puertos 2000 y 7949 son empleados para verificar la recepción de dicho acuse.

```
IP 192.168.1.1. 47578 > 192.168.1.2. 2000: UDP, length 123
IP 192.168.1.2. 2000 > 192.168.1.1. 33225: UDP, length 53
IP 192.168.1.1. 36347 > 192.168.1.2. 7949: UDP, length 163
IP 192.168.1.2. 7949 > 192.168.1.1. 36347: UDP, length 23
```

Así mismo, cuando se precise dificultar el análisis, existe la posibilidad de utilizar puertos aleatorios que permitan al destino el envío continuo de acusos de recibo a través de diferentes puertos.

```
IP 192.168.1.1. 47578 > 192.168.1.2. 2000: UDP, length 123
IP 192.168.1.2. 1280 > 192.168.1.1. 33225: UDP, length 53
IP 192.168.1.1. 36347 > 192.168.1.2. 7949: UDP, length 163
IP 192.168.1.2. 98634 > 192.168.1.1. 36347: UDP, length 23
```

La viabilidad de esta técnica se ha analizado mediante el estudio realizado en relación a los clientes más habituales de BitTorrent. Como se puede observar en la tabla 5.1, durante aproximadamente 4 minutos hemos capturado tráfico de dichas aplicaciones y hemos observado que para sesiones de duración muy corta el número de flujos generados en diferentes puertos es muy elevado. Por ello, en base a los resultados obtenidos y considerando además el caso BitComet en el que se superan los 10.0000 flujos en 4 minutos, resulta complicado analizar dichos flujos dada su elevada cantidad. Así mismo, también es necesario considerar que, como media, se generan 2628 puertos diferentes a disposición de los clientes BitTorrent.

Aplicación	Flujos	Puertos
BitComet	10890	8617
BitLord	890	587
Vuze	1704	1528
Azuerus	2512	2257
uTorrent	1865	1668
BitTorrent	1819	1112

Tabla 5.1: Puertos y flujos usados por BitTorrent

5.2.3. Payload noise

El algoritmo RC4 presenta una serie de vulnerabilidades [135, 136, 137] basadas en ataques de análisis de frecuencias, sin embargo este problema puede resolverse parcialmente mediante el empleo de la técnica de rotación de clave. El comportamiento de las mencionadas técnicas, reflejado en la figuras 5.4, 5.5 y 5.6, muestra la dispersión de frecuencias de paquetes generados por InCC sin ninguna firma generada en ellos. Los bytes están representados en el eje X y en

el eje Y está representado el número de ocurrencias del byte x. Del análisis de la figura 5.4, en la que se han cifrado 10.000 objetos utilizando la misma clave, se desprende que dada la escasez de distribución de frecuencias podría ejecutarse un ataque basado en las mismas. No obstante, como puede apreciarse en la figura 5.5, el empleo de claves aleatorias permite mejorar considerablemente la distribución de frecuencias con respecto al caso anterior.

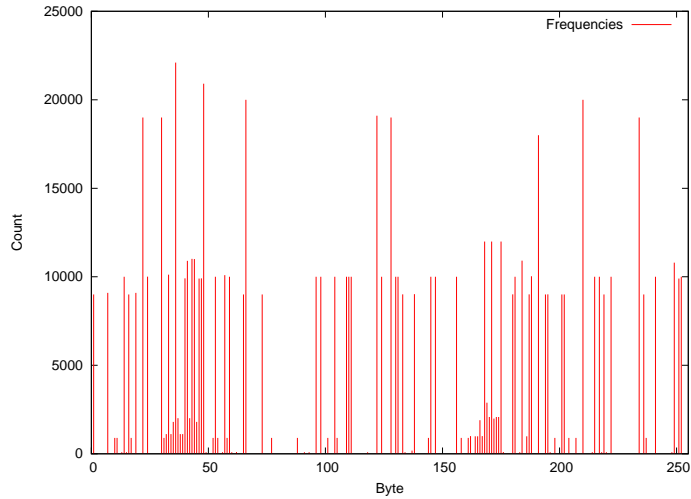


Figura 5.4: Cifrado de 10.000 objetos con RC4 [91]

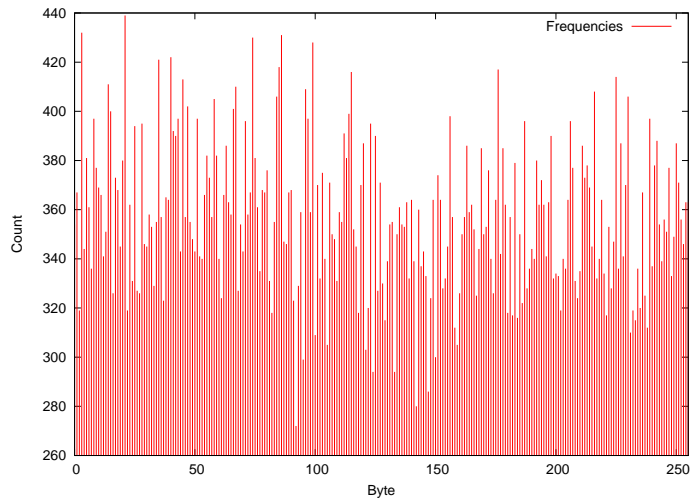


Figura 5.5: Cifrado de 10.000 objetos con RC4 mediante rotación de clave [91]

En la figura 5.6 se aprecia como InCC, en lugar de generar claves aleatorias en el cifrado, incorpora ruido que permite modificar el tamaño de los objetos de manera arbitraria. De esta forma se dificulta la detección, además de generar una distribución de frecuencias tan óptima como la que se obtendría mediante rotación de clave con RC4.

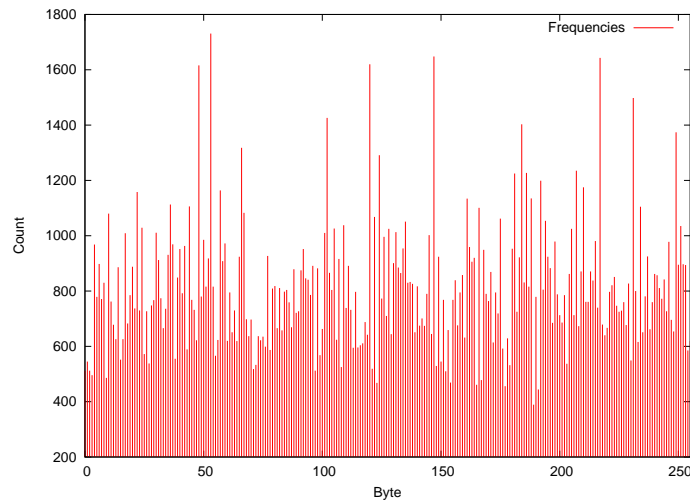


Figura 5.6: Cifrado de 10.000 objetos con RC4 incorporando ruido [91]

No obstante, cabe destacar que si bien la dispersión de frecuencias en las figuras 5.5 y 5.6 es similar, en términos de compresión no lo es como consecuencia del tamaño de los objetos que se están cifrando.

Este proceso lo denominamos 'Payload noise' y consiste en incorporar bytes aleatorios en partes de un objeto serializado, como podemos ver en la figura 5.7. El empleo de esta técnica permite conseguir la misma dispersión que con la rotación de clave (ver figuras 5.5 y 5.6).

Así mismo, esta estrategia también permite ajustar el tamaño de los paquetes a los detectados en la red. Por ejemplo, si el proceso AB detecta BitTorrent con tamaño de paquetes de 300 bytes, entonces InCC generará paquetes de tamaño similar al visto para camuflar los nuevos flujos en tráfico BitTorrent.

La figura 5.7 refleja el proceso de serialización del objeto y la forma de incorporar ruido a éste. En ella puede observarse como mediante la incorporación de clave y valores aleatorios es posible incrementar el tamaño del objeto, así como la dispersión de frecuencias que se generará.



Figura 5.7: Noise payload serialización [132]

5.2.4. IP randomness

Uno de los aspectos más interesantes del sistema consiste en que las direcciones IP que genera no pertenecen a ninguno de los sistemas que se quieren comunicar, como vemos en la figura 5.1. Esto es el resultado del aprendizaje las direcciones IP más empleadas en la red o de la configuración directa las direcciones IP falseadas utilizadas entre el origen y destino.

El procedimiento propuesto es el siguiente:

- Cuando el *Sender* y el *Receiver* de la figura 5.1 analizan el tráfico de la red todas las direcciones IP identificadas por medio de firmas de tráfico son almacenadas en una tabla temporal. Dichas direcciones IPs dependen de la duración de los flujos y del tipo de usuario. Por ejemplo, si un usuario tarda 20 minutos en realizar una descarga de un fichero con un cliente BitTorrent, la tabla temporal guardará el tiempo de duración de cada flujo que, en este caso, será de 20 minutos.
- En caso de que el nodo *Sender* envíe un mensaje al nodo *Receiver* éste tiene dos opciones: la primera consiste en utilizar estas direcciones temporales y la segunda se basa en emplear direcciones IP aleatorias de la subred en la que se detectó el flujo.

Si un atacante avanzado accede a uno de los Routers por el que tienen que pasar los mensajes, éstos pasarán totalmente desapercibidos ya que éste detectará una sesión BitTorrent y un elevado número de flujos de dicha sesión con direcciones IP legítimas. Es conveniente resaltar que una sesión P2P suele generar una media de 3.000 flujos aproximadamente.

La utilización de direcciones temporales tiene la ventaja, con respecto a las aleatorias, de que los flujos generados son idénticos a los detectados e incluyen direcciones IP que generan tráfico, dificultando el análisis de éstas.

En la figura 5.1, versión simplificada de la figura 2.1, los nodos *S* representan a los sistemas que necesitan intercambiar información (donde estaría InCC en el *Sender* y el *Receiver*), los nodos *R* son Routers y *A* y *B* simbolizan a dos usuarios intercambiando archivos mediante una aplicación P2P. Cuando *Sender* o *Receiver* detectan actividad P2P, éstos guardan las direcciones IP de *A* y de *B*, como se muestra en el punto t1 de la figura. Si la red está bien configurada, dichos nodos tendrán que detectar los mismos flujos de datos, ya que deben utilizar las mismas firmas de detección. Teniendo en cuenta esto, cuando el

Sender quiera enviar información al nodo *Receiver*, éste generará tráfico con las direcciones origen y destino de *A* y de *B*, como se observa en el punto t2. El nodo *Receiver* no reencaminará el datagrama generado por el origen ya que si llega al destino real, en nuestro caso *B*, se podrían generar datagramas ICMP que podrían delatar nuestra presencia (ver punto 7 en el apartado 5.2.1).

Una de las ventajas del empleo de direcciones IP temporales consiste en que los datagramas generados por el sistema pueden quedar totalmente camuflados en el tráfico detectado entre *A* y *B*. Esto aporta mayor robustez al sistema en comparación con las direcciones aleatorias, que incluso podrían carecer de sentido dentro de la red, ya que podría darse el caso en el que la dirección IP generada no formara parte de ningún Host de la red y un atacante avanzado pudiera reconocer este hecho por medio del empleo de técnicas de detección de puertos [138].

5.2.5. Signature poisoning

Los NIDS y herramientas como tcpdump [119] pueden ser utilizadas por parte de un atacante para la inspección el tráfico dentro de una red, de manera que el atacante puede identificar servidores HTTP, DNS, IMAP y STMP, aportando información sobre el modo de sincronizar un ataque desde el interior de la red.

Por medio de firmas de tráfico de red de diferentes herramientas [130, 131, 139], la librería puede ocultar los flujos generados en el tráfico de la red. El sistema implementa soporte UDP a modo de prueba de concepto con lo que cualquier tráfico UDP que disponga de firma de tráfico, como DNS, BitTorrent, Gnutella, Sopcast o tráfico de juegos, puede ser empleado por el sistema.

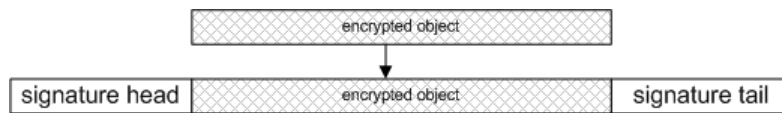


Figura 5.8: Signature poisoning [91]

La figura 5.8 muestra un mensaje cifrado con RC4 y el modo en que se le añadiría la firma, tanto en la cabecera como en la cola, en base a la firma proporcionada.

En la siguiente salida del comando tcpdump podemos ver un mensaje generado por el sistema. El paquete resultante contiene los bytes 0x474e y 0x4403, que corresponden a una firma del protocolo Gnutella, y el resto del mismo se encuentra cifrado mediante el algoritmo RC4.

```
IP 192.168.1.1.55728 > 192.168.1.2.4399: UDP, length 178
0x0000:  4500 00ce 2e5b 4000 4011 8870 c0a8 0101  E....[.@..p....
0x0010:  c0a8 0102 d9b0 112f 00ba 2044 474e 4403  ...../...GND.
0x0020:  578e df02 9088 bd1b e3db d268 5bf4 4ffc  W.....h[.0.
0x0030:  d626 4e10 9440 c93e c1a1 6249 ce2d 92df  .&N..@.>..bI.-..
```

5.2.6. Implementación

El sistema se compone de una librería multiplataforma con dos procesos claramente diferenciados, el *Application Behavior* (AB) y el *Obfuscator* (OB). Como podemos ver en la figura 5.9, el proceso AB es el encargado de analizar los flujos de la red e identificarlos utilizando firmas de tráfico que pueden ser importadas desde otras aplicaciones como Snort u OpenDPI.

Como podemos observar en el siguiente código 5.1, cuando un flujo no ha sido detectado y contiene información entonces, mediante la función *DTN_MatchSignatures* se comprueba si existe alguna firma de tráfico que lo identifique. Si existe firma de tráfico tanto origen como destino habrán detectado el flujo y, a continuación, se llamará a la función *INCC_ProcessIncomingFlow* que comprueba si es un flujo de detección o un flujo que contiene información.

```

if(flow->detected == 0){ // The flow is not detected
    if(segment_size > 0 ){ // the packet have payload
        signature = DTTN_MatchSignatures( _inccEngine->detect ,
            flow ,
            PKCX_GetPayload() ,
            segment_size );
        if(signature) {
            LOG(INCCLOG_PRIORITY_INFO,
                " Detecting '%s' on flow [%s:%d:%d:%s:%d] flow (0x%x)
                ",
                signature->name,
                PKCX_GetSrcAddrDotNotation() ,
                PKCX_GetSrcPort() ,
                protocol ,
                PKCX_GetDstAddrDotNotation() ,
                PKCX_GetDstPort() ,
                flow );

            INCC_ProcessIncomingFlow(signature , flow ,
                PKCX_GetPayload() , segment_size );
        }
    }
}

```

Código C 5.1: Procesamiento de paquetes entrantes de InCC

Una vez que AB identifica los flujos, éste guarda las direcciones IPs y la distribución de tamaño del paquete que identificó la firma. Seguidamente, AB notifica a OB que se ha identificado un protocolo y que el canal estaría listo para poder enviar un mensaje, de forma que el OB pueda utilizar la distribución del paquete detectado y la firma que se identificó para enviar tráfico al destino.

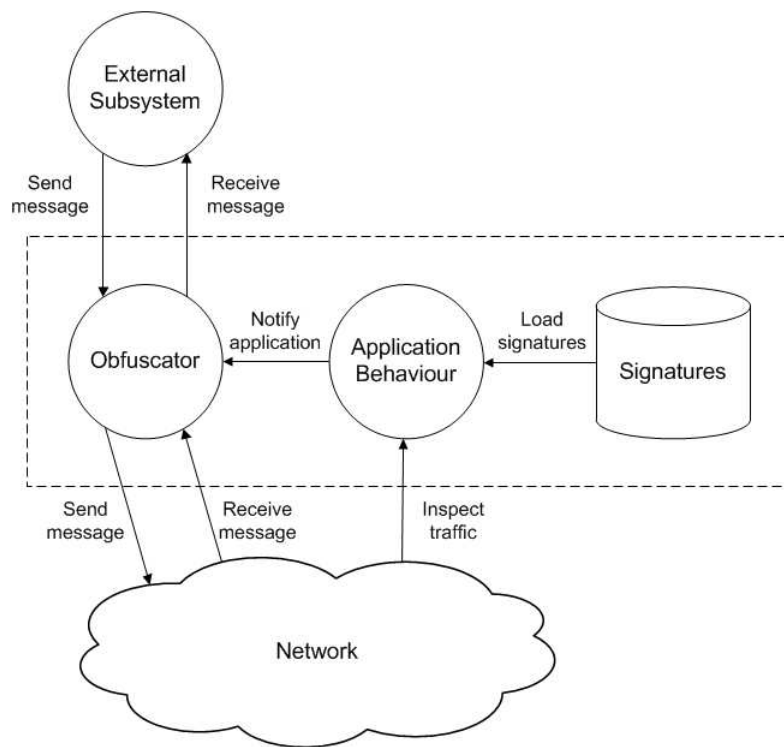


Figura 5.9: Modelo de procesos de InCC [91]

En la figura 5.10 se muestra el modo en que se genera el mensaje final, cuyo procedimiento es el siguiente:

1. Incorporar ruido al mensaje, tal y como se ha descrito en el apartado 5.2.3.
2. Seleccionar un puerto aleatorio para los siguientes paquetes, ver apartado 5.2.2.
3. Incluir la siguiente clave rotatoria para el cifrado, que sería la rotación de clave, ver apartado 5.2.3.
4. Cifrar la totalidad del contenido del mensaje con RC4.
5. Añadir firmas, tanto en la cabeza como en la cola del mensaje, ver apartado 5.2.5.

Una vez generado el mensaje, éste se enviará por la red de manera que solo pueda ser descifrado por el destino.

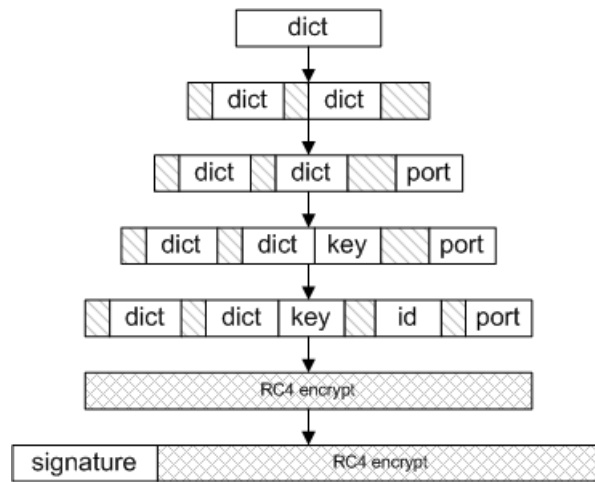


Figura 5.10: Proceso completo de un flujo InCC [91]

Teniendo en cuenta las opciones mencionadas, los flujos generados podrán camuflarse perfectamente en el tráfico detectado, de manera que los subsistemas externos podrán intercambiar información a través de las mismas redes sin comprometer su existencia en ellas. Nuestro método no modifica los paquetes, como ocurre con Xinwen [87], ni depende de protocolos específicos como Lucena [85] y, por ello, es válido para cualquier tráfico UDP e independiente del protocolo, lo que aporta mayor versatilidad en la creación de canales encubiertos. El sistema presenta únicamente la dependencia de las firmas de tráfico, sin embargo el elevado número de proyectos Open Source que existen en el mercado permiten tener acceso a éstas [130, 131, 139].

5.3. Experimentos y resultados

En este apartado vamos a probar y evaluar el sistema con tráfico real, de forma que podamos comprobar con datos reales si los mensajes generados por InCC quedan realmente ocultos en una red. Para verificar la invisibilidad de los mensajes de InCC utilizaremos OpenDPI [131] como herramienta de identificación de tráfico. Para la realización de los test hemos utilizado dos PCs Linux con kernels 2.6.38, uno de ellos con CPU Intel core duo 3.16GHz y el otro con una CPU Intel core duo de 2GHz. En primer lugar, se ha generado tráfico Gnutella en uno de los PCs. Hemos elegido las firmas de Gnutella, a modo de prueba de concepto, por simplicidad de uso al descargar un archivo y, seguidamente, hemos capturado la descarga de un fichero por parte de Gnutella en un fichero pcap. A continuación, con la finalidad de que el sistema pueda aprender el tráfico que hay en la red, hemos inyectado tanto en el origen como en el destino dicho tráfico, de manera que los dos sistemas detecten Gnutella, y finalmente hemos enviado un mensaje de InCC por la red.

Dado que el sistema es una aplicación híbrida (núcleo desarrollado en C e interfaz en Python), éste puede ser configurado según interese en cada caso específico. Concretamente, en la propuesta que nos ocupa, se ha aplicado la configuración que se muestra a continuación en el código fuente 5.2:

```

if __name__ == '__main__':

    parser = parseOptions()
    (options, args) = parser.parse_args()
    if (options.interface == None):
        parser.error("Argument is required")
        sys.exit(1)

    incc.INCP_Init()

    if (options.destination != None):
        incc.INCP_SetDestinationIP(options.destination)

    if (options.source != None):
        incc.INCP_SetSourceIP(options.source)

    if (options.timetolive != None):
        incc.INCP_SetPacketTTL(int(options.timetolive))

    if (options.show_payload != None):
        incc.INCP_ShowGeneratedPayload(options.show_payload)

    # TODO: the First key interchange could be done by using RSA
    # and with standar sockets and once the interchange have done
    # close the open socket and use random key rc4 for
    # the main encryption.
    if (options.rc4key != None):
        incc.INCP_SetEncryptionKey(options.rc4key)

    incc.INCP_SetSource(options.interface)

    # netbios specification:
    # first two bytes from transaction id
    # and the rest the signature candidate
    # Example of packet chaining
    # incc.INCP_AddSignature(1, "netbios",
    #     "^.{2}\\x01\\x10\\x00\\x01.*\\x00\\x20\\x00\\x01",
    #     "myhead", 6, "mytail", 6)
    # incc.INCP_AddSignature(2, "mypacket", "^myhead.*mytail",
    #     "myhead", 6, "mytail", 6)

    # A simple distributed tables of bittorrent signature.
    # incc.INCP_AddSignature(3, "torrent", "d1:ad2:id20:",
    #     "d1:ad2:id20:", 12, None, 0)

    # A detection of bittorrent and generate battlefield traffic
    # incc.INCP_AddSignature(3, "torrent", "d1:ad2:id20:",
    #     "\x11\x20\x00\x01\x00\x00\x50\xb9\x10\x11", 10, None, 0)

    incc.INCP_AddSignature(1, "gnutella", "^\\x47\\x4e\\x33\\x03",
        "\\x47\\x4e\\x33\\x03", 4, None, 0)

```



```

incc.INCP_Start()

try:
    incc.INCP_Run()
except (KeyboardInterrupt, SystemExit):
    incc.INCP_Stop()

if (options.statistics):
    incc.INCP_Stats()
incc.INCP_StopAndExit()

```

Código Python 5.2: Configuración de InCC

En la configuración mostrada se cargan en memoria, en primer lugar, las estructuras internas mediante la función *incc.INCP_Init()*. Seguidamente se configuran las diferentes funciones recibidas por comando, siendo la más relevante de todas ellas la denominada *incc.INCP_SetSource()*, que constituye una selección del dispositivo de red para el análisis del tráfico. A continuación, mediante la función *incc.INCP_AddSignature()* se procede con la definición de las firmas de tráfico a emplear en la que es necesario resaltar que si bien en el ejemplo de la configuración 5.2, únicamente se han definido algunas de ellas, el modelo propuesto es susceptible de ser ampliado con la finalidad de permitir la lectura de las firmas de un fichero o de bases de datos, entre otras. Finalmente, mediante la función *incc.INCP_Run()*, se pasa a un bucle activo en el que se realiza la mayor parte del trabajo.

Es importante destacar que el mediante el empleo de la red simplificada de la figura 2.1 es posible reducir los costes de implementación de la solución completa, ya que el despliegue de un entorno como el descrito en la figura tendría un elevado coste. No obstante, tal y como muestra la figura 5.1, las pruebas propuestas para la obtención de los resultados esperados en la prueba concepto pueden ser perfectamente simuladas en un entorno simplificado.

Posteriormente hemos fusionado el tráfico generado por el sistema con el de Gnutella, anteriormente capturado. Seguidamente, mediante herramientas como Snort [130] y OpenDPI [131] analizamos el fichero pcap resultante, de manera que los flujos del sistema propuesto queden completamente ocultos a las mismas.

La tabla 5.2, muestra la detección de tráfico de Gnutella con OpenDPI, incluyendo el número de bytes, paquetes y flujos detectados, y la tabla 5.3 contiene el tráfico de Gnutella, junto con el nuevo flujo que simula al de Gnutella. Es conveniente destacar que el flujo generado por el sistema propuesto contiene 51 paquetes y 13.827 bytes, lo que se supone una cantidad suficiente para reportar incidentes de seguridad.

Para verificar el rendimiento del sistema, concretamente la carga que supone la librería, ver figura 5.11, hemos generado 10.000 objetos con las diferentes opciones del sistema, descritas en los apartados 5.2.2, 5.2.3, 5.2.4, y 5.2.5. Los resultados muestran un consumo despreciable del sistema, aún disponiendo de todas sus opciones activas.

Protocolo	Paquetes	Bytes	Flujos
Unknown	61215	19031577	17
Dns	10	960	3
Http	115	44107	11
Netbios	53	7679	3
Gnutella	102656	30658221	270
Icmp	563	52454	21

Tabla 5.2: Detección de tráfico Gnutella con OpenDPI

Protocolo	Paquetes	Bytes	Flujos
Unknown	61215	19031577	17
Dns	10	960	3
Http	115	44107	11
Netbios	53	7679	3
Gnutella	102707	30672048	271
Icmp	563	52454	21

Tabla 5.3: Detección de tráfico de Gnutella e InCC con OpenDPI

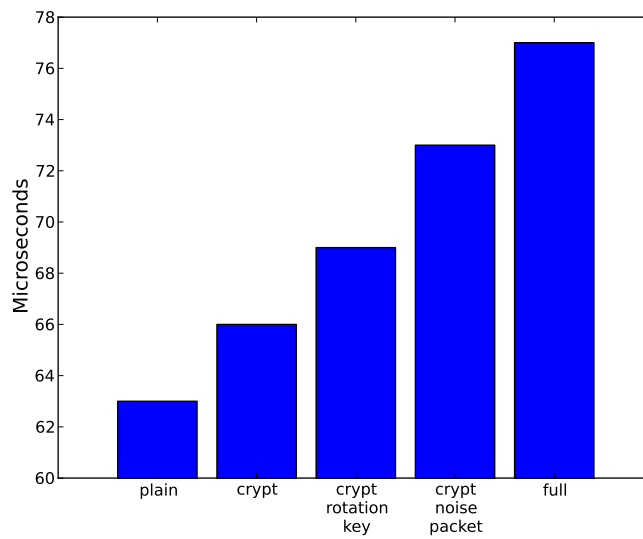


Figura 5.11: Rendimiento de las opciones de InCC [91]

5.4. Conclusiones

Hemos presentado una librería multiplataforma implementada en Python [140] y C para canales encubiertos, capaz de comunicarse con otros sistemas de manera totalmente transparente, sin delatar al origen ni al destino. Mediante el empleo de las opciones comentadas, en los apartados 5.2.2, 5.2.3, 5.2.4, y 5.2.5, se genera un sistema totalmente indetectable y modulable, como se muestra en los resultados del apartado 5.3.

El trabajo presentado ha sido verificado mediante herramientas externas, Snort y OpenDPI, que han comprobado la efectividad de los métodos propuestos, haciendo de InCC un canal totalmente invisible ante cualquier análisis de tráfico actual. De esta forma, se propone una nueva técnica capaz de evadir cualquier tipo de detección, camuflando nuestros flujos en los existentes en la red. InCC fue diseñado para soportar tráfico UDP, sin embargo realizando algunas modificaciones, la viabilidad de hacerlo para TCP también sería válida.

Muchas aplicaciones P2P envían tráfico basura con la finalidad de dificultar los análisis de red por parte de los ISPs. En este sentido, una posible ampliación del sistema propuesto consistiría en enviar tráfico basura mientras se envía el mensaje al destino. De esta manera, en caso de que el sistema no pudiera detectar tráfico en el que camuflarse, una buena solución consistiría en que los administradores de las redes dispusieran clientes P2P que, indirectamente, facilitarían el camuflaje del sistema en dicho tipo de tráfico.

Capítulo 6

Mitigación del DDoS

6.1. Planteamiento del problema

Descripción del problema

Mientras se realiza un ataque de denegación de servicio se saturan los recursos de la red y del sistema al que se ataca. Normalmente la mayoría de los ataques van dirigidos a infraestructuras web, de manera que un ataque es considerado satisfactorio si usuarios que no forman parte de dicho ataque, normalmente usuarios habituales del servicio, no pueden acceder a los recursos web.

Problemas existentes

El principal problema radica en poder ofrecer servicio a los usuarios que son legítimos de los que no lo son durante un ataque. Cabe recordar que un ataque DDoS tiene por objeto la saturación de los recursos, de manera que éstos queden inutilizados mientras se produce el mismo.

Solución propuesta

El sistema que proponemos, denominado DLimitter, aborda el problema desde la perspectiva de control de flujo. Para ello se procura tomar el control de aquellos flujos dirigidos a la infraestructura a proteger mediante una solución que controle la frecuencia de los paquetes, junto con otros mecanismos que mencionaremos en los siguientes apartados.

En el apartado 6.2 exponemos una descripción de la solución propuesta en la que se detallan y explican sus diferentes componentes, seguidamente en el apartado 6.3 mostramos la implementación del sistema, a continuación en el apartado 6.4 mostramos los resultados y los experimentos llevados a cabo y, finalmente, en el apartado 6.5 presentamos las conclusiones del sistema propuesto.

6.2. Solución

Con la finalidad de abordar la denegación de servicio, se propone una solución capaz de encaminar los paquetes IP del usuario a un sistema de colas, según su comportamiento. De esta forma, si un usuario es identificado por el sistema como peligroso o potencialmente peligroso, DLimiter dispondrá sus paquetes IP en una cola más restrictiva, con mayor retardo y con un flujo más lento. Por el contrario, si se trata de un usuario legítimo, sus paquetes serán gestionados mediante una cola con muy poco retardo, de manera que no afecte al usuario. De esta forma, en caso de producirse un ataque, el sistema propuesto será capaz de garantizar el servicio a los usuarios legítimos, sin que éstos sufran ningún corte o perturbación en el servicio.

6.2.1. Funcionalidad software

Como se muestra en la figura 6.1, el software DLimiter presenta una estructura basada en hilos que cooperan entre sí, en un sistema de colas y en estructuras internas que interactúan sobre los flujos de datos que circulan hacia los servidores web.

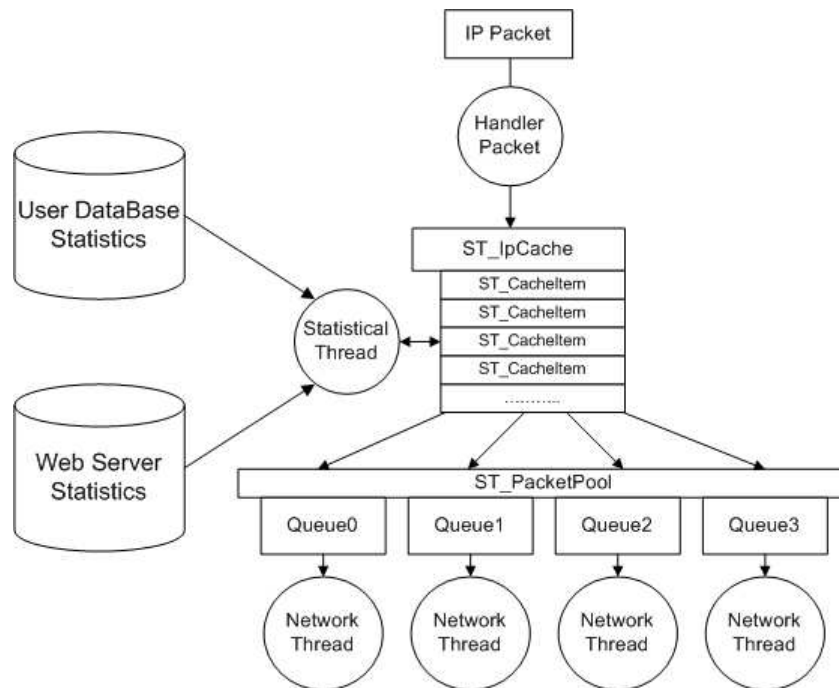


Figura 6.1: Modelo de hilos y datos [94]

El funcionamiento se describe a continuación. Inicialmente todos los flujos de un usuario se dirigirán a la misma cola y con idéntico retardo, no obstante,

en caso de identificar a un usuario sospechoso que consumiera mayor ancho de banda de lo normal, sus flujos serían trasladados a una cola con mayor retraso que la anterior. Por otra parte, los flujos de aquellos usuarios que realicen un uso correcto de los recursos se depositarán en una cola con menor retraso. Si bien no es propósito del sistema decidir si un usuario es legítimo o no, con la finalidad de desarrollar completamente el sistema se propone la utilización de un hilo estadístico denominado *Statistical Thread*. Este hilo será el responsable de gestionar los cambios de cola en base al consumo de ancho de banda de los usuarios, penalizando o gratificando su valor.

Así mismo, el sistema ofrece otra interesante funcionalidad que consiste en la utilización del marcado de usuario externo para determinar si un usuario es malicioso o no, facilitando con ello la integración del DLimitter en otros sistemas externos de seguridad, como listas de acceso, NIDS, Firewalls u otros.

DLimitter implementa un sistema de disciplina multicolos también para minimizar los falsos positivos y negativos provocados por una incorrecta decisión de los sistemas de detección, tal y como puede verse en la figura 6.2. De forma que si los paquetes IP de un usuario son trasladados a una cola con mayor retraso, y dicho retraso no es muy abrupto, el usuario prácticamente no notará ningún efecto en sus conexiones. Por ejemplo, si un usuario está constantemente demandando recursos del servidor, entonces se le moverá a una cola más restrictiva, con un retraso mayor en los paquetes IP del flujo. Consecuentemente, únicamente los usuarios que consuman más recursos serán movidos a colas con mayores retrasos.

Finalmente conviene resaltar que, de idéntico modo que Labrea [43], nuestro sistema incorpora un mecanismo de control sobre los flujos TCP que, mediante el empleo de control de ventana deslizante de TCP [141], permite ralentizar los flujos de determinados usuarios. Esta herramienta se emplea cuando las colas están llenas y en el nodo no existan más recursos de memoria disponibles para el almacenaje de paquetes IP, así como en aquellos otros casos en los que la CPU se encuentre saturada debido a un efecto Slashdot [142].

6.2.2. Características software

Con la finalidad de dar respuesta a las funcionalidades descritas en el apartado anterior, hemos desarrollado un software que reúne las siguientes características:

- El sistema puede disponerse en cualquier lugar de la red, aunque preferentemente éste se situará próximo a los servidores web.
- El sistema funciona como un Proxy transparente de forma que no pueda ser direccionable a nivel IP desde ningún lugar.
- En caso de saturación del sistema, como por ejemplo un efecto Slashdot, este debería disponer de mecanismos como el descarte de paquetes IP o el control de flujo, TCP como realizan muchos Honeypots [43].

- El software implementa un sistema que permite recibir información, facilitando la interacción de Dlimiter con otros sistemas externos como Firewalls, NIDS, Antivirus, etc... e incrementando, de este modo, la escalabilidad a la hora de desplegar la solución propuesta.
- El sistema de colas debe ser configurable y debe poder ajustarse dinámicamente, dependiendo de las condiciones de la red. Las principales disciplinas de colas soportadas son lineares y exponenciales. Las lineales se utilizarán durante un ataque y las exponenciales se usarán cuando no tengamos suficientemente certeza. La elección de las dos disciplinas ha sido tomada debido a que la naturaleza de un ataque DDoS es exponencial, donde miles de máquinas atacan una infraestructura.

6.3. Implementación

Dlimiter está basado en la arquitectura Netfilter[106] de Linux y constituye una aplicación multihilo que presenta las siguientes funcionalidades:

- Un hilo principal, denominado *HandlerPacket*, responsable de la distribución de los paquetes entrantes dependiendo del número de CPUs y de colas del sistema. De forma que, cuando un nuevo paquete IP llega al sistema, el *HandlerPacket* realizará una operación hash [143] sobre la dirección IP origen para acceder a una tabla hash, denominada *ST_IpCache* (ver código 6.1), en $O(1)$.

```

struct ST_CacheItem {
    int used; /* 0 = not used, 1 = used, 2 = used with
              colision */
    int type;
    time_t timestamp;
    int cachetype;
    int hash;
    int queue_destination;
    struct ST_IpInfo infoday[24];
    struct ST_IpInfo infohour[60];
    char ipaddress[MAX_CACHE_IP_DATA];
    struct ST_CacheItem *next;
} __attribute__((packed));

typedef struct ST_CacheItem ST_CacheItem;

struct ST_IpCache {
    unsigned long items;
    unsigned long colisions;
    int itemsbytype[THRUST_LEVEL_MAX];
    ST_CacheItem cache[CACHE_PRIME];
};

typedef struct ST_IpCache ST_IpCache;

```

Código C 6.1: Estructura de tabla hash

En aquellos casos en los que la *ST_IpCache* estuviera llena al tener ocupados la totalidad de sus slots, el *HandlerPacket* comenzaría a descartar los paquetes IP entrantes que no puedan ser gestionados. En el caso que nos ocupa, la *ST_IpCache* ha sido dimensionada para más de 31.000 usuarios, con una memoria de aproximadamente 61MBytes, cantidad más que suficiente para el modelo propuesto. Dentro de esta cache tenemos *ST_CacheItem* que contiene información relevante del usuario como la dirección IP, el identificador de destino de la cola a donde van los paquetes, el estado de los flujos, contadores de bytes transmitidos por día hora y minuto, valor máximo de cuota, etc...

- El sistema emplea datos de los usuarios como son la media de flujos, peticiones, bytes transmitidos y desviaciones estándar, basándonos en trabajos anteriores [100, 105, 144, 145]. Mediante el empleo de dicha información, hemos implementado un hilo estadístico que permite colocar en colas superiores o inferiores a los usuarios, además de controlar el consumo de la CPU, evitando de este modo su colapso, incluso en el caso de que se sature la red antes que los servidores. Es necesario resaltar que en este tipo de ataques todos los segmentos TCP SYN serán descartados automáticamente y que no es el objetivo de *DLimiter* realizar una detección del ataque. Sin embargo en ausencia de un sistema externo de identificación que indique a *DLimiter* qué usuarios penalizar, el sistema emplearía dicha información estadística para, al menos, penalizar o gratificar a los usuarios.

En relación con los falsos positivos, como puede observarse en la figura 6.2, aunque un usuario haya sido marcado como atacante y, por ello, transferido a una cola superior más restrictiva, si durante un período de tiempo éste presentara un comportamiento estadístico adecuado, el hilo podría corregir la decisión adoptada y trasladarle a una cola menos restrictiva.

Por otra parte, como puede observarse en la parte inferior de la figura 6.2, para la gestión de los falsos negativos tendríamos el caso contrario al anterior. Así, en el caso de un usuario que estuviera llevando a cabo un ataque, y éste fuera considerado como válido por un sistema externo, el hilo estadístico comprobaría constantemente las estadísticas, de manera que éste podría ser penalizado, disminuyendo el número de peticiones por minuto que podría llevar a cabo.

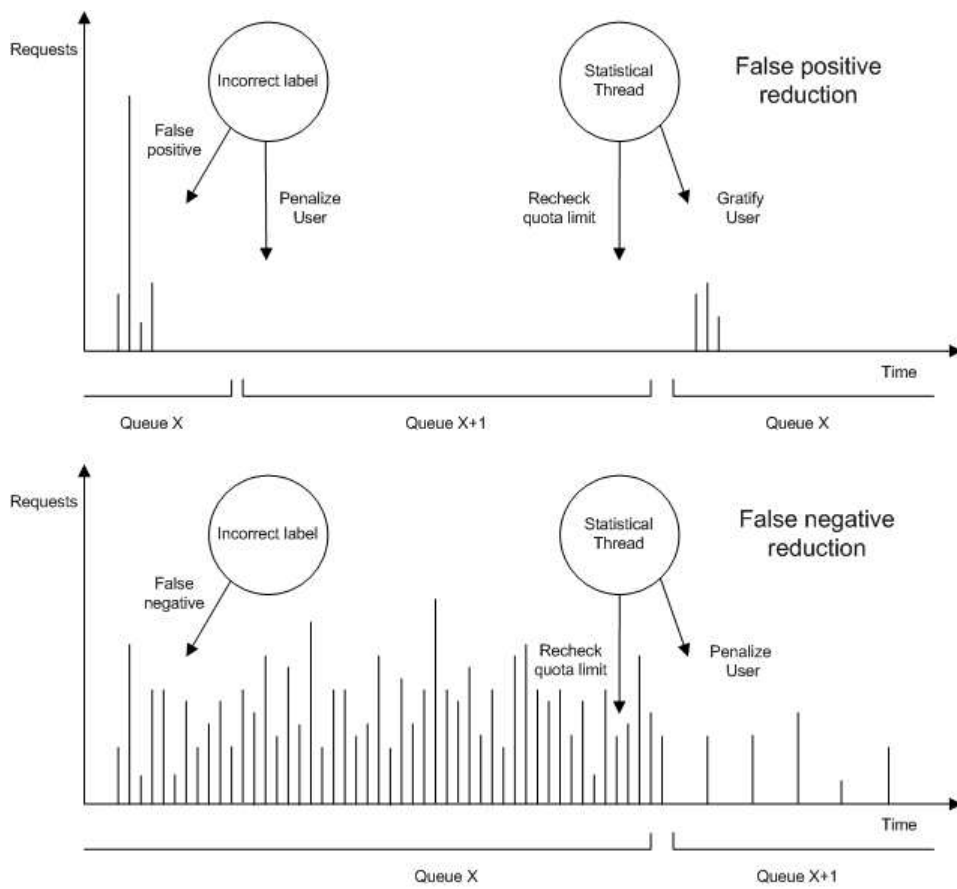


Figura 6.2: Reducción de falsos positivos y negativos [94]

- Disponemos también de un pool de hilos, al que denominamos *ST_ThreadPool* (ver código 6.2), en el que el número de hilos dependerá del número de CPUs y del número de colas que seleccionemos. En dicho pool cada hilo recibe una estructura *ST_ThreadInfo*, que contiene el número de cola en el que debe actuar y que denominamos *Poolnumber*.

```

/** Struct for Thread Information */
struct ST_ThreadInfo {
    int used;
    int priority;
    int policy;
    int poolnumber;
    int cpunumber;
    void (*ThreadFunction)(void *thread_info);
} __attribute__((packed));

typedef struct ST_ThreadInfo ST_ThreadInfo;
    
```

```
ST_ThreadInfo ST_ThreadPool[MAX_THREADS];
```

Código C 6.2: Información de los hilos

Con el identificador de cola del sistema, el hilo ya puede acceder a las estructuras de colas, descritas en el código 6.3, que contienen la siguiente información relevante: una lista enlazada de paquetes donde cada paquete es un *ST_PacketNode*, el tiempo de espera de cada cola, el número de hilos adjuntados a la cola, el máximo límite de congestión, etc...

A su vez, cada cola se encuentra definida en la estructura *ST_PacketListHeader*, que incluye información relativa al número de hilos de cada cola, número total de paquetes IP de la cola, mutex para la gestión de la concurrencia entre hilos, etc....

```
struct ST_PacketNode {
    unsigned long id;
    unsigned int hook;
    struct timeval timestamp;
    unsigned char *payload;
    struct ST_PacketNode *next;
    struct ST_PacketNode *ant;
} __attribute__((packed));

typedef struct ST_PacketNode ST_PacketNode;

struct ST_PacketListHeader {
    int used;
    ST_PacketNode *start;
    ST_PacketNode *end;
    pthread_mutex_t mutex;
    pthread_cond_t cond;
    unsigned long items;
    unsigned long totalitems;
    unsigned long threadsdelay;
    int queuetype;
    int workingthreads;
    unsigned long ips_on_pool;
    int peakload;
    int action; /* NF_ACCEPT, NF_DROP */
    ST_PoolStat stats;
    void (*AddPacket) (int poolnumber, ST_PacketNode * pkt);
    void (*GetPacket) (int poolnumber, ST_PacketNode ** ptk);
};

typedef struct ST_PacketListHeader ST_PacketListHeader;
```

Código C 6.3: Estructuras de colas y paquetes

Como se muestra en el código fuente 6.4, cada uno de los hilos que se ejecuta sobre una cola lleva a cabo las siguientes operaciones:

1. Se asigna un hilo a cada CPU mediante la primitiva *pthread_set_affinity_np* [146]. Esta primitiva del sistema operativo permite fijar el hilo de eje-

cución a cierta CPU, de manera que éste no pueda ser ejecutado en otra CPU.

2. A continuación el hilo recibirá la información de la cola que tiene que gestionar, para poder así manejar los paquetes IP de dicha cola.

Un vez inicializado el hilo, éste pasa a un bucle infinito en el que se llevan a cabo siguientes operaciones:

1. Se inspecciona y actualiza el tiempo de espera del primer paquete IP de la cola.
2. Si el tiempo de espera del paquete es igual o mayor que el tiempo de espera de la cola, entonces el paquete es liberado de la cola y se encamina. Esta operación es repetida hasta que el tiempo de espera del último paquete de la cola sea menor que el tiempo de espera de la cola.
3. En caso de que el tiempo de espera no haya expirado, el hilo actualiza el tiempo del paquete y suspende su ejecución mediante la función Posix *usleep*. El referido tiempo de espera del hilo depende del tiempo de espera de la cola, del tiempo que le falte al último paquete en la cola y de un valor aleatorio previamente calculado.

A continuación podemos ver en el código 6.4 los pasos anteriormente explicados en forma de algoritmo en el lenguaje C.

```

while (TRUE) {
    rand_val = UT_GetRandom (2,10);

    PL_LockPacketPool (threadinfo->poolnumber);
reload:
    PL_InspectPacket (threadinfo->poolnumber,&pkt);
    if (pkt != NULL) {
        gettimeofday (&current_time ,NULL);
        delay_time.tv_sec = 0;
        delay_time.tv_usec = queuedelay;

        UT_TimevalAdd(&on_buffer_time ,
            &pkt->packet_time ,
            &delay_time);

        if (timercmp(&current_time,&on_buffer_time,>)) {
            /* the thread should forward the packet */
            PL_GetPacketFromTailPool (threadinfo->poolnumber ,
                &pkt);
            if (pkt != NULL) {
                /* release the packet and free it */
                PK_SetPacketResolution (pkt ,
                    pool [threadinfo->poolnumber] . action);

                PL_FreePacketNode (pkt);
            }
            goto reload;
        }
    }
}

```

```

    }
}

PL_UnlockPacketPool(threadinfo->poolnumber);
/* Pool Sleeping */
usleep(queuedelay / rand_val);
}

```

Codigo C 6.4: Código hilo

Las diferentes colas, tiempos de espera e hilos se configuran en un fichero Python. En nuestra investigación hemos realizado tests con distribuciones lineales y exponenciales, pero el modelo es genérico y se podrían utilizar distribuciones de tiempo diferentes como, por ejemplo, distribuciones cuadráticas.

Seguidamente, en el código 6.5, vemos un ejemplo de configuración enviada por D-Bus al sistema. En ella podemos observar que se crean cuatro colas con una distribución exponencial.

```

bus = dbus.SessionBus()
try:
    s = bus.get_object('dlimiter.engine', '/dlimiter/engine')
except:
    print "No Filter engine available on the bus"
    sys.exit(-1)

iface = dbus.Interface(s, dbus_interface='dlimiter.engine')

# Now configure all the queues

iface.SetQueueDelay(0,10)

iface.CreateQueue()
iface.SetQueueDelay(1,100)

iface.CreateQueue()
iface.SetQueueDelay(2,1000)

iface.CreateQueue()
iface.SetQueueDelay(3,10000)

iface.CreateQueue()
iface.SetQueueDelay(4,100000)

```

Codigo Python 6.5: Código configuración DLimitter

6.4. Experimentos y resultados

En los siguientes apartados se procede a explicar los experimentos llevados a cabo para evaluar el funcionamiento del sistema propuesto en una amplia variedad de configuraciones posibles.

Para la simulación de un ataque DDoS hemos utilizado Siege [10] como principal herramienta. Entre las diversas opciones que ésta ofrece se encuentra el

número de clientes que realizarán las peticiones HTTP. En el caso concreto que nos ocupa, y con la finalidad de saturar tanto la red como el servidor, hemos establecido dicho valor en 50, disponiendo por tanto de 50 clientes HTTP lanzando peticiones contra un servidor web. Hemos utilizado dicho valor porque con él, tanto la red como el servidor entran en saturación, objetivo que precisamente perseguimos en nuestros experimentos.

Descripción de los test.- Uno de los principales problemas a la hora de simular un ataque DDoS consiste en la dificultad para acceder a trazas reales, ya que las compañías atacadas no desean ver dañada ni su credibilidad, ni su imagen como consecuencia de dichos ataques. Otro factor condicionante lo constituyen las restricciones legales existentes a la hora de difundir dicha información.

Con la finalidad de verificar el funcionamiento de nuestra implementación, hemos realizado un prototipo que implementa todas las funcionalidades comentadas en el apartado 6.3.

Si bien en el mercado existen otras herramientas para generar tráfico, como Loic [11] o Slowloris [5, 7], en la simulación del ataque DDoS hemos optado por Siege, herramienta recientemente empleada en diversos ataques, que ofrece las mismas funcionalidades que las anteriores y que destaca por su facilidad de uso y por las opciones que dispone.

En la simulación también hemos utilizado tres PCs que envían diferente tráfico al servidor (ver figura 6.3). Cada uno de ellos envía peticiones HTTP, que incluyen a su vez 50 procesos concurrentes cada uno. De esta manera se crean 150 procesos que envían peticiones al servidor web objeto del ataque.

Así mismo, para la simulación del ataque se proponen seis tests, tres de ellos lineales y otros tres exponenciales, que se detallan a continuación en los apartados 6.4.1 y 6.4.2. Si bien cada uno de ellos envía el mismo tráfico, en cada test se modifican las configuraciones del sistema, tales como el número de colas, su distribución, etc...

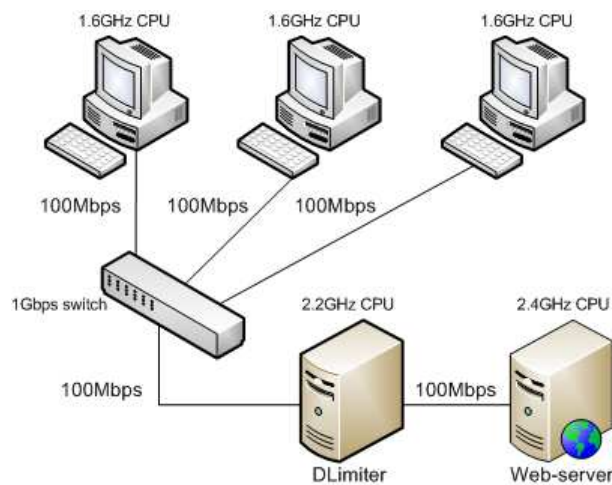


Figura 6.3: Arquitectura hardware [94]

Para la realización de los test se han utilizado tres tipos de usuario modelo, cada uno de ellos caracterizado en uno de los PCs:

- El primer usuario corresponde al 'usuario desconocido de gran demanda' (*unknown high-demanding user*), al que denominamos *uhd-user*, y representa un Bot HTTP que envía peticiones tan rápido como le sea posible. El PC en el que se instala este tipo de usuario simboliza a los usuarios que forman parte del ataque.
- El segundo tipo de usuario es el llamado 'usuario desconocido de demanda media' (*unknown medium-demanding user*), también denominado *umd-user*, que es similar al anterior solo que la frecuencia de peticiones es justo la mitad.
- El tercer tipo de usuario es el llamado 'usuario habitual' (*regular-user*), denominado *r-user*, y simboliza a aquellos usuarios que hacen un uso correcto de los recursos del servidor web.

Los usuarios *uhd-users* y *r-users* se configuran con la misma frecuencia de acceso, es decir con las mismas peticiones HTTP por segundo, y tratan de utilizar todo el ancho de banda disponible. Los *umd-users* utilizan un rate igual que el anterior pero con un valor aleatorio entre cada una de las peticiones entre 0 y 1 segundo, de manera que su tasa de envío es mucho menor que los anteriores. En nuestro caso utilizamos todo el ancho de banda de la red que soporta nuestra red LAN (100Mb).

Durante un tiempo aproximado de 10 minutos y basándonos en la arquitectura hardware propuesta de la figura 6.3, en cada una de las máquinas se han lanzado 50 clientes concurrentes que acceden a las diferentes páginas web del servidor. A su vez, con la finalidad de estudiar los efectos sobre el sistema y

sobre los usuarios, tales como el tipo de disciplina de cola y el número de colas instanciadas, hemos realizado seis test diferentes.

Como ya se ha mencionado anteriormente, de los seis tests realizados, tres de ellos corresponden a colas de distribución lineal (apartado 6.4.1) y los otros tres, al mismo experimento con colas de distribución exponencial (apartado 6.4.2). Estos test se han realizado utilizando un servidor Apache HTTP 2.0 en un servidor 2.4GHz Pentium generico PC y los procesos de Siege se han ejecutado en maquinas 1.6GHz Pentium genérico, todos ellos conectados a una red Ethernet a 100Mbps y con un Switch también a 100Mbps. Así mismo, con la intención de crear un servidor web más realista, en él se ha desplegado un servicio Gotocode [147] el cual utiliza una base de datos Mysql 5.0, de manera que el servidor web no simplemente sirve paginas web, si no que tiene mucha más funcionalidad. De esta forma el servidor web es mucho más realista.

6.4.1. Colas lineales

Cuatro colas lineales.- Como podemos ver en la figura 6.4, concretamente en la parte inferior de la misma, aproximadamente en el segundo 120 los r-users toman el control de la totalidad del ancho de banda disponible, mientras que los otros usuarios mantienen el mismo ancho de banda durante el resto del experimento. Como consecuencia de ello, podemos concluir que prácticamente toda la operación de gestión de ancho de banda, así como de cambio de colas, se produce en los primeros 120 segundos, de manera que durante un ataque en tan solo dos minutos los r-users podrían disponer de la totalidad del ancho de banda.

Ocho colas lineales.- Durante este experimento podemos comprobar que en la parte media de la figura 6.4, se observa una menor tendencia que en el caso anterior. Concretamente, si comparamos este caso con el anterior nos daremos cuenta de que los r-users dispondrán de todo el ancho de banda a partir del segundo 300. Por ello, podemos concluir que, si bien en este caso el cambio de ancho de banda es menos abrupto que en el experimento anterior, también es menos efectivo. Resulta igualmente interesante observar cómo durante los 100 primeros segundos los r-users y los uhd-users disponen del mismo ancho de banda.

Doce colas lineales.- Como puede observarse en la parte superior de la figura 6.4, en este caso la distribución es similar a la anterior excepto en que los r-users tardan un poco más en tener todo el ancho de banda disponible. Desde el punto de vista de los r-users, éste es el peor caso, ya que durante los primeros minutos del ataques se sufren diversos retrasos al no poder disponer de suficiente ancho de banda. Por ello, podemos concluir que a mayor número de colas se ofrece peor servicio a los r-users.

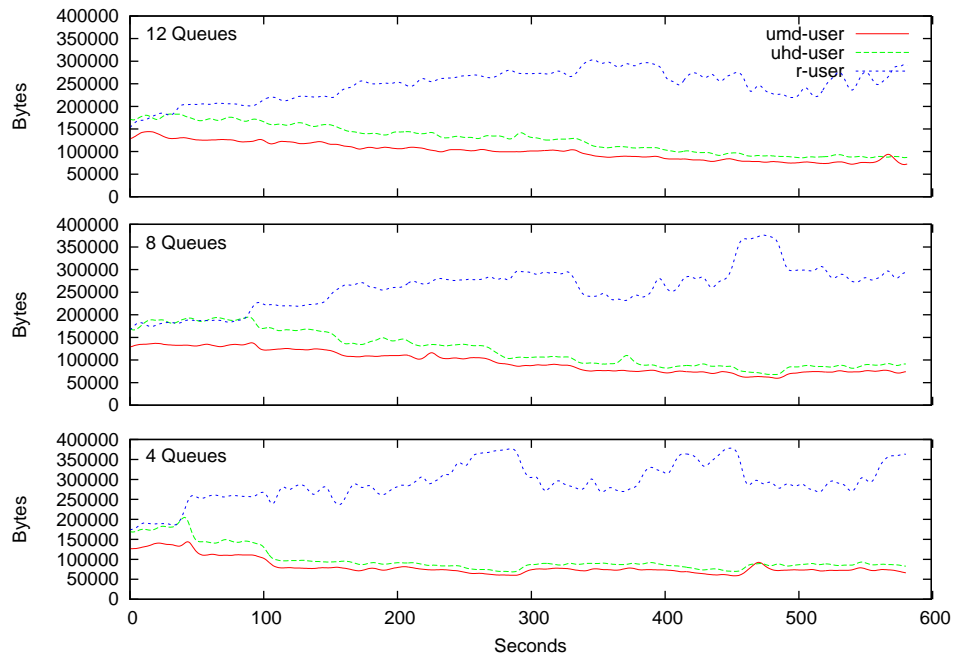


Figura 6.4: Distribución de ancho de banda con colas lineales [94]

En base a los resultados obtenidos, podemos concluir que el tiempo de respuesta de los usuarios desconocidos decrece conjuntamente con el número de colas empleadas, lo que provoca que los usuarios habituales (r-users) se beneficien de este efecto. Como consecuencia de ello y de la necesidad de provocar un gran cambio en la distribución del ancho de banda, al inicio del ataque se utilizará un número reducido de colas.

En la tabla 6.1, que muestra los diferentes resultados obtenidos en los tests realizados, se observa el número de paquetes IP gestionados por el sistema, así como el número de peticiones realizadas por cada uno de los usuarios, la tasa de peticiones y el tiempo de respuesta del servidor en segundos. De los resultados obtenidos se desprende que los r-users disminuyen su tasa de peticiones y los uhd-user la incrementan cuando se incrementa el número de colas.

Usuarios	Colas	Paquetes IP	Peticiones	Tasa de peticiones	Tiempo de respuesta (segundos)
umd-user	4	224.162	20.110	33,5	1,97
uhd-user	4	273.860	24.459	40,76	1,95
r-user	4	783.807	72.915	121,58	0,32
umd-user	8	263.245	24.071	40,12	1,48
uhd-user	8	335.953	30.597	51,02	1,45
r-user	8	701.142	65.259	108,72	0,42
umd-user	12	273.365	25.109	41,82	1,38
uhd-user	12	348.544	31.999	53,31	1,37
r-user	12	664.599	61.925	103,14	0,47

Tabla 6.1: Resultados lineales

6.4.2. Colas exponenciales

Cuatro colas exponenciales.- Como se puede apreciar en la parte inferior de la figura 6.5, en este caso se observa un cambio abrupto a partir del segundo 140. Este resultado es muy parecido al de las cuatro colas lineales, con la salvedad de que durante los primeros 120 segundos los r-users experimentan una calidad de servicio considerablemente inferior a la ofrecida en las colas lineales.

Ocho colas exponenciales.- La parte media de la figura 6.5 muestra como los r-users obtienen el ancho de banda disponible a partir del segundo 400 y que, durante este período de tiempo, todos los usuarios comparten el ancho de banda. Así mismo, tal y como se desprende de los resultados de la tabla 6.2, la tasa de peticiones para los r-users con ocho colas es mayor que respecto a la de cuatro. Esto nos lleva a considerar que un incremento en el número de colas exponenciales no genera ninguna ganancia para los r-users, sino todo lo contrario.

Doce colas exponenciales.- Tal y como se muestra en la parte superior de la figura 6.5, este caso puede ser considerado como el más desfavorable de los propuestos ya que los r-users obtienen el ancho de banda justo al final del experimento, sin existir apenas gestión de colas. Cabe también resaltar que este caso también es el peor en términos de tiempos de respuesta, como se muestra en la tabla 6.2.

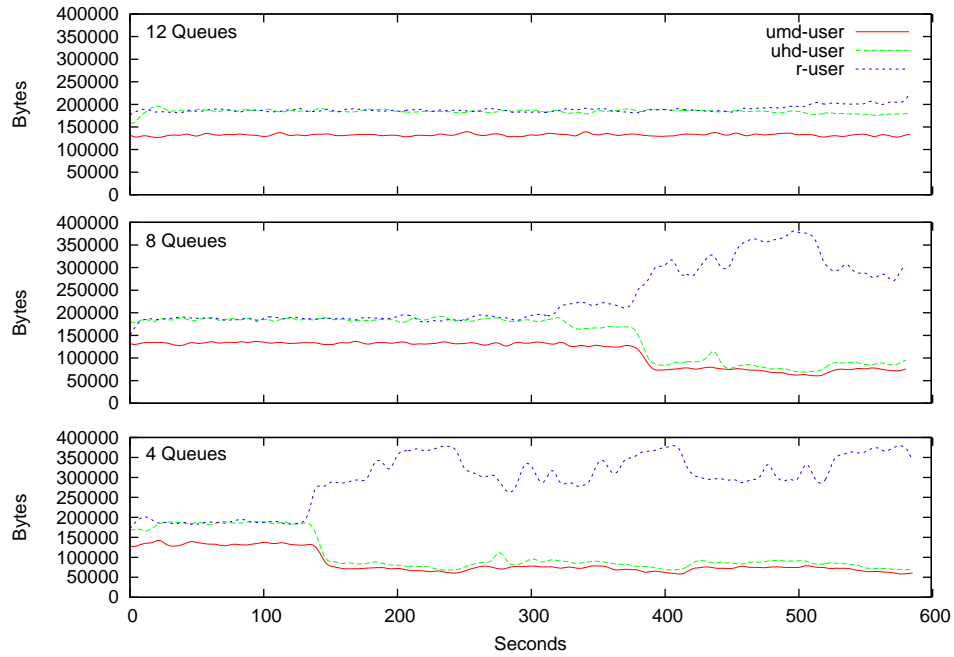


Figura 6.5: Distribución de ancho de banda con colas exponenciales [94]

Usuarios	Colas	Paquetes IP	Peticiones	Tasa de peticiones	Tiempo de respuesta (segundos)
umd-user	4	236.454	21.435	35,74	1,78
uhd-user	4	295.869	26.618	44,4	1,75
r-user	4	781.977	72.624	121,1	0,32
umd-user	8	303.998	27.976	46,64	1,13
uhd-user	8	404.422	37.174	61,91	1,11
r-user	8	633.493	58.854	98,15	0,52
umd-user	12	359.689	33.492	55,85	0,78
uhd-user	12	499.513	46.474	77,4	0,79
r-user	12	515.127	48.056	80,08	0,75

Tabla 6.2: Resultados exponenciales

Del análisis realizado se desprende que en la gestión con colas exponenciales los r-users obtienen un importante incremento en el ancho de banda, pero con mayor lentitud que con la gestión de colas lineales. Por otra parte, los usuarios desconocidos experimentan el efecto contrario y el sistema reduce su ancho de banda con mayor rapidez que con las colas lineales.

Por tanto, en vista de los resultados obtenidos en los apartados 6.4.1 y 6.4.2 de esta investigación, podemos afirmar que las colas lineales presentan una mejor reacción al principio del ataque. En la figura 6.6, se pueden observar

las diferentes latencias aplicadas a las colas durante los tests y como, debido a la naturaleza del comportamiento lineal, éste penaliza más rápidamente a los usuarios desconocidos que el comportamiento exponencial.

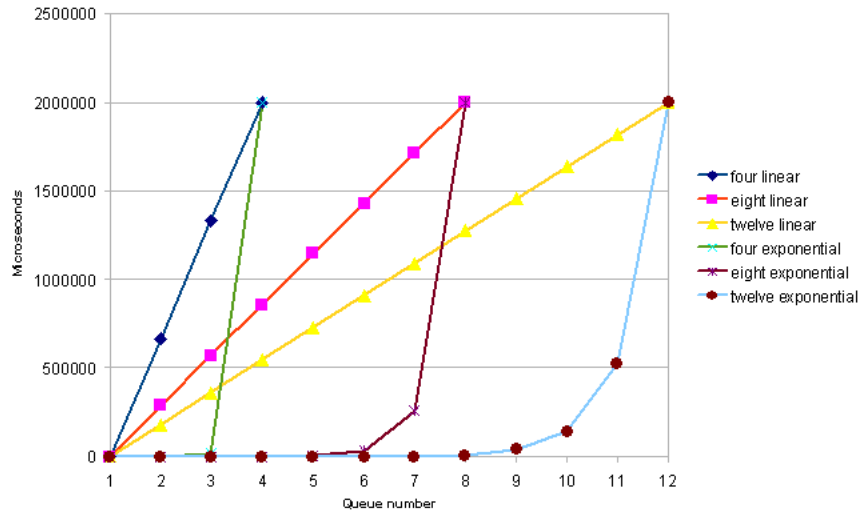


Figura 6.6: Configuración de las latencias de las colas [94]

En el caso de que un ataque sea detectado con colas lineales, los *uhd-users* se verán afectados por degradado de sus conexiones TCP y los *r-users* mantendrán una latencia inferior que los que iniciaron el ataque. Por otra parte, si el ataque tiene lugar con colas exponenciales, el retraso entre colas es despreciable, los usuarios tendrán las mismas latencias y compartirán el ancho de banda, con la obvia degradación de servicio de los *r-users*.

Así mismo, cuando no se produce ningún ataque se recomienda el empleo de colas exponenciales ya que los usuarios podrían compartir el ancho de banda disponible. Este caso resulta especialmente útil cuando disponemos de usuarios cuyas conexiones TCP es conveniente penalizar ya que, si bien éstos no alteran el servicio, consumen demasiado ancho de banda.

Por ello, la utilización de ambas disciplinas, dependiendo de si se produce un ataque o no y de la duración del mismo, aportan mayor versatilidad al sistema.

6.5. Conclusiones

Se ha presentado y realizado un ataque DDoS simulado en condiciones de laboratorio para cuyo fin hemos utilizado tres maquinas generadoras de tráfico, con un total de 150 clientes lanzando peticiones HTTP concurrentemente contra un único servidor web y con unos ratios muy elevados, tal y como puede observarse en los resultados anteriores.

Durante la simulación del ataque hemos realizado seis tests probando las diferentes combinaciones entre las colas lineales y las colas exponenciales y hemos analizado el comportamiento en los diferentes tipo de usuarios que hemos caracterizado.

Por otra parte, resulta muy complejo detectar y mitigar un ataque DDoS debido a las dificultades técnicas para realizar una simulación lo más realista posible. En este sentido es necesario destacar que el mayor ataque DDoS [113] realizado del que se tiene constancia ha sido de 300 GBs por segundo, lo que nos aporta una cifra exacta y un marco de referencia de la dificultad para disponer de un entorno de red que nos permita simular dicha cantidad de tráfico.

El sistema propuesto, denominado DLimitter, tiene un comportamiento esperado e incorpora monitorización de usuario para corregir falsos positivos y negativos de sistemas exteriores. Así mismo, la QoS de los usuarios habituales no se degrada durante un ataque de estas características, que es el objetivo principal.

Como se refleja en los resultados incluidos en el apartado 6.4, un sistema multi-cola de bajo coste es capaz de mitigar el efecto de un ataque y de reducir su impacto en los usuarios habituales, además de poder modificar dinámicamente el número de colas utilizadas en base a los parámetros externos al sistema.

Mediante los mecanismos ya expuestos, el software mostrado ha permitido gestionar más de 3.868.677 paquetes IP en los experimentos de colas lineales y 4.030.542 en las colas exponenciales, además de gestionar de manera satisfactoria más de 729.137 peticiones HTTP.

De los resultados citados se desprende que mediante la solución propuesta, de bajo coste y con un hardware no excesivamente caro, es posible mitigar de manera satisfactoria un ataque DDoS.

Capítulo 7

Conclusiones

Como ya se ha expuesto anteriormente la presente tesis tiene por objeto el desarrollo de una propuesta compuesta por tres sistemas, detección, comunicación y mitigación, que cooperan entre sí con la finalidad de ofrecer una adecuada protección frente al tan habitual problema de la denegación de servicio que tiene lugar en Internet y que afecta a todo tipo de infraestructuras.

La solución propuesta se compone del sistema de detección, que es responsable de la identificación de los posibles ataques a las infraestructuras de servidores, del sistema de mitigación que es el encargado de controlar el flujo de los usuarios a los servidores y, finalmente, del sistema de comunicación que permite la comunicación de los dos sistemas anteriores con otros y también con ellos mismos, de manera que dichas comunicaciones no pueden ser identificadas ni alteradas.

Dichos sistemas son totalmente independientes y, por ello, pueden integrarse o desplegarse en cualquier nodo de la red y distribuirse del modo más conveniente. Así mismo, al tratarse de sistemas completamente seguros, éstos no son accesibles a nivel IP por ningún elemento de la red, salvo que éstos se encontraran aislados por una red física de gestión.

Cabe resaltar que la arquitectura distribuida para la detección, comunicación y mitigación de la denegación de servicio a nivel de aplicación que se aporta puede ser implantada en una instalación con hardware limitado, es configurable e integrable y su difusión de código garantiza el cumplimiento de la licencia GPL [148], de manera que puede ser descargada y utilizada por otros investigadores.

A su vez, la utilización del sistema de intercomunicación D-Bus [116] permite integrar los diferentes procesos mediante el sistema de comunicación más estándar empleado en Linux, además de facilitar la monitorización de los mismos de una manera sencilla e independiente.

La arquitectura con servicios que se considera puede ser integrada en otros servicios externos, como Routers, Firewalls o NIDS, de manera rápida y eficaz, lo que dota a la solución de gran versatilidad.

Así mismo, se aporta un sistema escalable en el que, dependiendo de la topología de la red y de los servicios a proteger, éste puede adaptarse a las

diversas necesidades de la red, especialmente en términos económicos.

A modo de resumen final, se aportan a continuación las principales conclusiones extraídas de lo expuesto en los capítulos anteriores que, a su vez, permiten demostrar que los sistemas propuestos han sido capaces de detectar, comunicar y mitigar ataques DDoS mediante el empleo de recursos con un hardware limitado.

Conclusiones sobre la detección Para la detección de un ataque DDoS hemos empleado un sistema software basado en el análisis del tráfico y en la generación de un modelo de tráfico lo más similar posible al de los servidores web a proteger.

Para llevar a cabo dicha tarea hemos considerado la utilización de un algoritmo que cuenta con tres fases claramente diferenciadas que, al mismo tiempo, nos ha permitido detectar cualquier actividad anómala llevada a cabo, como por ejemplo una auditoría ilegal realizada contra un servidor web público de Internet. Concretamente, tal y como se ha demostrado en el apartado 4.3, tanto los resultados aportados por el algoritmo como el consumo de CPU han sido satisfactorios, lo que permite resaltar el elevado grado de robustez del sistema propuesto, que incluso nos conduce a pensar que podría ser empleado en un entorno más agresivo y con mayor número de ataques como, por ejemplo, una intrusión a una gran infraestructura Cloud. Por otra parte, el algoritmo que se ha propuesto ha sido capaz de gestionar los diferentes caminos, URLs y estadísticas de más de 14.000 usuarios de tráfico real de Internet en tiempo real.

Para la gestión de los falsos positivos y falsos negativos generados en la detección se analizaron las trazas de tráfico de un servidor multidominio público Español. Si bien inicialmente consideramos dichas trazas de tráfico como cero falsos positivos, ya que los servidores disponían de sistemas comerciales de seguridad, nuestro algoritmo fue capaz de descubrir sin ningún tipo de intervención actividad del Bot de Google, así como una intrusión ilegal que no había podido ser detectada por los sistemas de seguridad existentes.

Conclusiones sobre la comunicación Con la finalidad de aportar un sistema de comunicaciones completamente transparente ante cualquier ataque de red hemos creado InCC (*Invisible Cover Channel*), un canal encubierto de almacenamiento, independiente del tráfico existente en la red, capaz de generar tráfico de red idéntico al detectado que, además, permite el camuflaje de la información en tráfico VoIP, juegos, P2P o cualquier otro tipo que disponga de firmas.

Para generar dicho sistema, y que éste pueda pasar totalmente desapercibido, se ha considerado la combinación de las siguientes técnicas: en primer lugar, el empleo de direcciones IP origen y destino falseadas que no dispongan de direccionamiento alguno; en segundo lugar, la utilización de puertos aleatorios como los de las aplicaciones P2P que permitan el continuo envío de información a diferentes puertos; en tercer lugar, el cifrado de la información mediante el algoritmo RC4 con las variantes expuestas y, finalmente, para la identificación

de dichas firmas se propone la detección del tráfico, la generación del mismo y el empleo de firmas de tráfico de otros proyectos ya existentes, como pueden ser Snort [130], Bro [57] u otros analizadores de red.

Tal y como se desprende de los resultados obtenidos, la combinación de las técnicas expuestas convierte a InCC en un canal robusto, prácticamente invisible ante cualquier análisis de tráfico.

Conclusiones sobre la mitigación Para poder mitigar los efectos de un ataque DDoS hemos desarrollado un sistema denominado DLimitter, que se basa en la utilización totalmente configurable de las colas de los paquetes IP, tanto en lo relativo a su número, como al tipo de retraso.

Para los experimentos realizados hemos considerado el empleo de distribuciones lineales y exponenciales y, en base al tipo de distribución aplicada, se aplican latencias a las diferentes colas del sistema. De este modo, la utilización, al inicio de un ataque DDoS, de una distribución lineal resulta más adecuada y, en ausencia de dicha intrusión, aporta mejores resultados una distribución exponencial. En cualquier caso, el sistema propuesto tiene la capacidad de modificar su comportamiento en tiempo de ejecución, aportando de esta manera mayor flexibilidad ante un ataque.

DLimitter cuenta también con un sistema de gestión automática de penalización que, ante la ausencia de un sistema de detección externo, le permite penalizar o gratificar automáticamente a los usuarios, además de poder reducir el número de falsos positivos generados por un sistema de detección.

7.1. Aportaciones de la tesis

Entre las aportaciones de la presente tesis cabe destacar el desarrollo de tres sistemas novedosos que cooperan entre sí con la finalidad de paliar los efectos de un ataque DDoS.

Dichos sistemas se encuentran validados por la comunidad científica y están disponibles en Internet bajo licencia GPL, de manera que cualquier persona interesada puede continuar con dichos estudios.

Adjuntamos, a continuación, las direcciones web que contienen el código fuente de los diferentes elementos de esta tesis:

- Sistema de mitigación <https://bitbucket.org/camp0/dlimiter>.
- Sistema de comunicación <https://github.com/camp0/incc>.
- Sistema de detección <http://code.google.com/p/polyvaccine>.

En este sentido conviene destacar que la solución propuesta constituye una excepción ya que gran parte de los estudios existentes no proporcionan código fuente, se basan en simulaciones teóricas y carecen de evaluaciones realizadas con tráfico real.

Lo expuesto en la presente tesis funciona bajo cualquier sistema Linux y, si se quisiera, podría utilizarse en sistemas comerciales con la finalidad de garantizar la protección de aquellos usuarios de medianas empresas que no pueden asumir los costes de sistemas de grandes marcas.

Por otra parte, la flexibilidad de integración que ofrece el sistema permite su implantación con otras soluciones existentes, como por ejemplo servidores web, Firewalls, NIDS o Antivirus, facilitando la realización de extensiones en el mismo.

7.2. Publicaciones

Mencionamos, a continuación, las publicaciones en las que se basa la presente tesis doctoral:

Luis Campo-Giralte, Cristina Conde, Isaac Martín de Diego, and Enrique Cabello. Detecting denial of service by modelling web-server behaviour. *Computers & Electrical Engineering*, 39(7):2252–2262, 2013

Luis Campo Giralte, Cristina Conde, Isaac Martin de Diego, and Enrique Cabello. Incc: Hiding information by mimicking traffic in network flows. In *10th International Conference on Security and Cryptography (SECRYPT 2013)*, Reykjavik, Iceland, July 2013. SciTePress

Luis Campo Giralte, Cristina Conde, Isaac Martin de Diego, and Enrique Cabello. Dlimiter : Mitigating distributed denial of service by using multiple discipline queues. In *International Conference on Mechatronics*,

Electronics and Automotive Engineering, Cuernavaca, Mexico, November 2013. IEEE

Luis Campo Giralte, Cristina Conde, Isaac Martín de Diego, and Enrique Cabello. Incc: Evading interception and inspection by mimicking traffic in network flows. In *International Joint Conference on e-business and Telecommunications*, Lecture Notes in Computer Science. Springer-Verlag, 2013

7.3. Trabajos futuros

Otra posible utilidad del sistema propuesto, a desarrollar en un futuro, consiste en la implementación de los estudios realizados en plataformas de *Voice over IP* (VoIP), de manera que dichas infraestructuras pudieran ser protegidas de una manera segura.

En la actualidad se está desplegando la tecnología *Long Term Evolution* (LTE), también denominada 4G, en la que los terminales móviles soportarán el protocolo SIP [149] (*Session Initiation Protocol*). Como consecuencia de ello, todo apunta a que en un futuro próximo, las antiguas centralitas telefónicas pasarán a ser servidores SIP y, por tanto, también serán susceptibles de verse afectadas por ataques DDoS, lo que implicaría un elevado sobrecoste para un considerable número de operadores móviles.

En este sentido, nuestro modelo es perfectamente extrapolable a las infraestructuras SIP ya que, como ocurre con HTTP, su protocolo se basa en un texto legible. Por ello, la realización de nuevos experimentos, análisis de trazas y reutilización del código, sería posible ampliar las aplicaciones del modelo que hemos propuesto.

Bibliografía

- [1] David Dittrich. The "tribe flood network" distributed denial of service attack tool. <http://staff.washington.edu/dittrich/misc/tfn.analysis.txt>, 1999.
- [2] G.R. Zargar and P. Kabiri. Identification of effective network features to detect smurf attacks. In *Research and Development (SCOReD), 2009 IEEE Student Conference on*, pages 49–52, 2009.
- [3] Jason Barlow and Woody Thrower. Tfn2k - an analysis. http://packetstormsecurity.org/files/view/10135/TFN2k_Analysis-1.3.txt, 2000.
- [4] David Dittrich. The "stacheldraht" distributed denial of service attack tool. <http://staff.washington.edu/dittrich/misc/stacheldraht.analysis.txt>, 1999.
- [5] Slowloris http dos. <http://ha.ckers.org/slowloris/>, 2013.
- [6] Mitigating slowloris. http://www.cert.org/blogs/vuls/2009/07/slowloris_vs_your_webserver.html, 2009.
- [7] Slowloris and iranian ddos attacks. <http://isc.sans.edu/diary.html?storyid=6622>, 2012.
- [8] Hypertext transfer protocol – http/1.1. <http://www.ietf.org/rfc/rfc2616.txt>, 2013.
- [9] Hulk, web server dos tool. <http://www.sectorix.com/2012/05/17/hulk-web-server-dos-tool>, 2013.
- [10] Siege. <http://www.joedog.org/index/siege-home>, 2013.
- [11] Low orbit ion cannon. <http://sourceforge.net/projects/loic/>, 2013.
- [12] Alert (ta12-024a) 'anonymous' ddos activity. <http://www.us-cert.gov/cas/techalerts/TA12-024A.html>, 2012.
- [13] Arbor networks. <http://www.arbornetworks.com/>, 2013.

- [14] [dns-operations] lots of queries for txt records? <https://lists.dns-oarc.net/pipermail/dns-operations/2009-April/003779.html>, 2009.
- [15] Dns attack downs internet in parts of china. <http://www.networkworld.com/news/2009/052109-dns-attack-downs-internet-in.html>, 2009.
- [16] The imddos botnet: Discovery and analysis. http://www.damballa.com/downloads/r_pubs/Damballa_Report_IMDDOS.pdf, 2010.
- [17] Media temple network unavailability. <http://status.mediatemple.net/weblog/category/system-incidents/1319-mt-media-temple-network-unavailability/>, 2010.
- [18] Wikileaks using amazon servers after attack. <http://blogs.wsj.com/digits/2010/11/29/wikileaks-using-amazon-servers-after-attack/>, 2010.
- [19] Wikileaks and the failure of cyberattacks as censorship. <http://blogs.forbes.com/andygreenberg/2010/11/28/wikileaks-and-the-failure-of-cyberattacks-as-censorship/>.
- [20] 'operation payback' attacks target mastercard and paypal sites to avenge wikileaks. <http://thelede.blogs.nytimes.com/2010/12/08/operation-payback-targets-mastercard-and-paypal-sites-to-avenge-wikileaks/>, 2010.
- [21] Operation payback. http://en.wikipedia.org/wiki/Operation_Payback.
- [22] Prolexic attack report. <http://www.prolexic.com/ddos-attack-reports/index.html>.
- [23] David Moore, Colleen Shannon, Douglas J. Brown, Geoffrey M. Voelker, and Stefan Savage. Inferring internet denial-of-service activity. *ACM Trans. Comput. Syst.*, 24(2):115–139, 2006.
- [24] The anatomy of an anonymous attack. <http://www.imperva.com/download.asp?id=312>, 2011.
- [25] Imperva's web application attack report. <http://www.imperva.com/download.asp?id=344>, 2012.
- [26] William G.J. Halfond, Jeremy Viegas, and Alessandro Orso. A Classification of SQL-Injection Attacks and Countermeasures. In *Proc. of the International Symposium on Secure Software Engineering*, Mar. 2006.
- [27] Kasra Amirtahmasebi, Seyed Reza Jalalinia, and Saghar Khadem. A survey of sql injection defense mechanisms. In *ICITST*, pages 1–8. IEEE, 2009.

- [28] High orbits and slowlorises: understanding the anonymous attack tools. <http://arstechnica.com/business/news/2012/02/high-orbits-and-slowlorises-understanding-the-anonymous-attack-tools.ars>, 2012.
- [29] Threat: Pandora ddos toolkit. http://www.prolexic.com/kcresources/prolexic-threat-advisories/Prolexic-Threat-Advisory-PANDORA_08.08.12/index.html, 2012.
- [30] M.M. Andrade and N. Vlajic. Dirt jumper: A key player in today's botnet-for-ddos market. In *Internet Security (WorldCIS), 2012 World Congress on*, pages 239–244, 2012.
- [31] Hackmageddon.com. <http://hackmageddon.com/>, 2013.
- [32] Jelena Mirkovic and Peter Reiher. A taxonomy of ddos attack and ddos defense mechanisms. *ACM SIGCOMM Computer Communication Review*, 34:39–53, 2004.
- [33] How to handle millions of new tor clients. <https://blog.torproject.org/blog/how-to-handle-millions-new-tor-clients>, 2013.
- [34] D. Mankins, R. Krishnan, C. Boyd, J. Zao, and M. Frenzt. Mitigating distributed denial of service attacks with dynamic resource pricing. In *ACSAC '01: Proceedings of the 17th Annual Computer Security Applications Conference*, page 411, Washington, DC, USA, 2001. IEEE Computer Society.
- [35] Kihong Park and Heejo Lee. On the effectiveness of route-based packet filtering for distributed dos attack prevention in power-law internets. In *In Proc. ACM SIGCOMM*, pages 15–26, 2001.
- [36] P. Ferguson and D. Senie. Network ingress filtering: Defeating denial of service attacks which employ ip source address spoofing, 2000.
- [37] Unicast reverse path forwarding enhancements for the internet service provider. <http://www.cisco.com/web/about/security/intelligence/urpf.pdf>, 2013.
- [38] John Ioannidis and Steven M. Bellovin. Implementing pushback: Router-based defense against ddos attacks. In *In Proceedings of Network and Distributed System Security Symposium*, 2002.
- [39] Aman Garg and A L Narasimha Reddy. Mitigating denial of service attacks using qos regulation. In *In Proceedings of International Workshop on Quality of Service (IWQoS)*, pages 45–53, 2001.
- [40] Aman Garg and A. L. Narasimha Reddy. Mitigation of dos attacks through qos regulation. *Microprocessors and Microsystems*, 28(10):521–530, 2004.

- [41] The honeynet project. <http://www.honeynet.org/>, 2013.
- [42] Niels Provos. Developments of the honeyd virtual honeypot. <http://www.honeyd.org/>, 2013.
- [43] Tom Liston. Labrea: "sticky" honeypot and ids. <http://labrea.sourceforge.net/>, 2013.
- [44] Niels Provos. A virtual honeypot framework. In *Proceedings of the 13th conference on USENIX Security Symposium - Volume 13*, SSYM'04, pages 1–1, Berkeley, CA, USA, 2004. USENIX Association.
- [45] Sherif M. Khattab, Chatree Sangpachatanaruk, Daniel Moss, Rami Melhem, and Taieb Znati. Roaming honeypots for mitigating service-level denial-of-service attacks. In *Proceedings of the 24th International Conference on Distributed Computing Systems (ICDCS'04)*, ICDCS '04, pages 328–337, Washington, DC, USA, 2004. IEEE Computer Society.
- [46] Sherif Hazem Noureldin, Sherif Hashem, and Salma Abdalla. Computer forensics guidance model with cases study. In *Proceedings of the 2011 Third International Conference on Multimedia Information Networking and Security*, MINES '11, pages 564–571, Washington, DC, USA, 2011. IEEE Computer Society.
- [47] Gregory H. Carlton and Hill Zhou. A survey of cloud computing challenges from a digital forensics perspective. *IJITN*, 3(4):1–16, 2011.
- [48] Boldizar Bencsath and Istvan Vajda. Protection against ddos attacks based on traffic level measurements, 2004.
- [49] Yu Chen, Kai Hwang, and Wei-Shinn Ku. Collaborative detection of ddos attacks over multiple network domains. *IEEE Trans. Parallel Distrib. Syst.*, 18(12):1649–1662, 2007.
- [50] Anukool Lakhina, Mark Crovella, and Christophe Diot. Mining anomalies using traffic feature distributions. *SIGCOMM Comput. Commun. Rev.*, 35(4):217–228, 2005.
- [51] Kai Hwang, Min Cai, Ying Chen, and Min Qin. Hybrid intrusion detection with weighted signature generation over anomalous internet episodes. *IEEE Trans. Dependable Secur. Comput.*, 4(1):41–55, 2007.
- [52] Xin Liu, Xiaowei Yang, and Yanbin Lu. To filter or to authorize: network-layer dos defense against multimillion-node botnets. In *SIGCOMM '08: Proceedings of the ACM SIGCOMM 2008 conference on Data communication*, pages 195–206, New York, NY, USA, 2008. ACM.
- [53] Jelena Mirkovic, Alefiya Hussain, Brett Wilson, Sonia Fahmy, Peter Reiher, Roshan Thomas, Wei-Min Yao, and Stephen Schwab. Towards user-centric metrics for denial-of-service measurement. In *ExpCS '07: Proceedings of the 2007 workshop on Experimental computer science*, page 8, New York, NY, USA, 2007. ACM.

- [54] Dimitris Gavrilis, Ioannis S. Chatzis, and Evangelos Dermatas. Detection of web denial-of-service attacks using decoy hyperlinks.
- [55] Stefan Seufert and Darragh O'Brien. Machine learning for automatic defence against distributed denial of service attacks. In *Proceedings of IEEE International Conference on Communications, ICC 2007, Glasgow, Scotland, 24-28 June 2007*, pages 1217–1222, 2007.
- [56] Vern Paxson. Bro: a system for detecting network intruders in real-time. *Computer Networks*, 31(23-24):2435–2463, 1999.
- [57] The bro network security monitor. <http://www.bro.org/>, 2013.
- [58] Tom Callahan, Mark Allman, and Vern Paxson. A longitudinal view of http traffic. In Arvind Krishnamurthy and Bernhard Plattner, editors, *PAM*, volume 6032 of *Lecture Notes in Computer Science*, pages 222–231. Springer, 2010.
- [59] Ashley Chonka, Jaipal Singh, and Wanlei Zhou. Chaos theory based detection against network mimicking ddos attacks. *Comm. Letters.*, 13(9):717–719, 2009.
- [60] Lan Li and Gyungho Lee. Ddos attack detection and wavelets. *Telecommunication Systems*, 28(3-4):435–451, 2005.
- [61] Norman Danner, Danny Krizanc, and Marc Liberatore. Detecting denial of service attacks in tor. pages 273–284, 2009.
- [62] Roger Dingledine, Nick Mathewson, and Paul Syverson. Tor: The second-generation onion router. In *In Proceedings of the 13th USENIX Security Symposium*, pages 303–320, 2004.
- [63] Lan Wang, Qishi Wu, and Dung Dinh Luong. Engaging edge networks in preventing and mitigating undesirable network traffic. In *NPSEC '07: Proceedings of the 2007 3rd IEEE Workshop on Secure Network Protocols*, pages 1–6, Washington, DC, USA, 2007. IEEE Computer Society.
- [64] *Detection of HTTP-GET flood attack based on analysis of page access behavior*, pages 232 – 5. IEEE, 2007.
- [65] Christos Siaterlis and Vasilis Maglaris. Detecting incoming and outgoing ddos attacks at the edge using a single set of network characteristics. In *Proceedings of the 10th IEEE Symposium on Computers and Communications*, pages 469–475, Washington, DC, USA, 2005. IEEE Computer Society.
- [66] Luis Campo-Giralte, Cristina Conde, Isaac Martín de Diego, and Enrique Cabello. Detecting denial of service by modelling web-server behaviour. *Computers & Electrical Engineering*, 39(7):2252–2262, 2013.

- [67] The dimes project. <http://www.netdimes.org>, 2013.
- [68] Dshield, cooperative network security community. <http://www.dshield.org/>, 2013.
- [69] Brent Chun, Jason Lee, Hakim Weatherspoon, and Brent N. Chun. Net-bait: a distributed worm detection service. Technical report, 2002.
- [70] Niels Provos and Introduction To Steganography. Hide and seek: Introduction to steganography, 2003.
- [71] D. Kundur and K. Ahsan. Practical internet steganography: Data hiding in IP. In *Proc. Texas Workshop on Security of Information Systems*, College Station, Texas, April 2003.
- [72] K. Ahsan and D. Kundur. Practical data hiding in TCP/IP. In *Proc. Workshop on Multimedia Security at ACM Multimedia '02*, French Riviera, December 2002.
- [73] Bo Xu, Jia-zhen Wang, and De-yun Peng. Practical protocol steganography: Hiding data in ip header. In *AMS '07: Proceedings of the First Asia International Conference on Modelling & Simulation*, pages 584–588, Washington, DC, USA, 2007. IEEE Computer Society.
- [74] Eric Cole. *Hiding in Plain Sight: Steganography and the Art of Covert Communication*. John Wiley & Sons, Inc., New York, NY, USA, 1 edition, 2003.
- [75] David Llamas, Alan Miller, and Colin Allison. An evaluation framework for the analysis of covert channels in the tcp/ip protocol suite. In *ECIW*, pages 205–214. Academic Conferences Limited, Reading, UK, 2005.
- [76] Sebastian Zander, Grenville J. Armitage, and Philip Branch. A survey of covert channels and countermeasures in computer network protocols. *IEEE Communications Surveys and Tutorials*, 9(1-4):44–57, 2007.
- [77] Sarah H. Sellke, Chih-Chun Wang, Saurabh Bagchi, and Ness B. Shroff. Tcp/ip timing channels: Theory to implementation. In *INFOCOM*, pages 2204–2212. IEEE, 2009.
- [78] Emanuel P. Freire, Artur Ziviani, and Ronaldo M. Salles. On metrics to distinguish skype flows from http traffic. *J. Network Syst. Manage.*, 17(1-2):53–72, 2009.
- [79] Sebastian Zander, Grenville J. Armitage, and Philip Branch. An empirical evaluation of ip time to live covert channels. In *ICON*, pages 42–47. IEEE, 2007.
- [80] Lucas Nussbaum, Pierre Neyron, and Olivier Richard. On robust covert channels inside dns. In Dimitris Gritzalis and Javier Lopez, editors, *SEC*, volume 297 of *IFIP*, pages 51–62. Springer, 2009.

- [81] Xiapu Luo, Edmond W. W. Chan, and Rocky K. C. Chang. Clack: A network covert channel based on partial acknowledgment encoding. In *Proceedings of IEEE International Conference on Communications, ICC 2009, Dresden, Germany, 14-18 June 2009*, pages 1–5, 2009.
- [82] Chaim Sanders, Jacob Valletta, Bo Yuan, Daryl Johnson, and Peter Lutz. Employing entropy in the detection and monitoring of network covert channels. *The 2012 International Conference on Security and Management*, 2012.
- [83] Krzysztof Szczypiorski. Hiccups: Hidden communication system for corrupted networks. In *In Proc. of: The Tenth International Multi-Conference on Advanced Computer Systems ACS'2003, October 22-24, 2003 Miedzyzdroje*, pages 31–40, 2003.
- [84] Wojciech Mazurczyk and Krzysztof Szczypiorski. Steganography in handling oversized ip packets. *CoRR*, abs/0907.0313, 2009.
- [85] Norka B. Lucena, James Pease, Payman Yadollahpour, and Steve J. Chapin. Syntax and semantics-preserving application-layer protocol steganography. In *Proceedings of the 6th Information Hiding Workshop*, pages 164–169, 2004.
- [86] Sedat Akleyek, Levent Emmungil, and Urfat Nuriyev. A modified algorithm for peer-to-peer security. *Appl. Comput. Math.* 6, no.2, pages 258–264, 2007.
- [87] Xinwen Fu, Yong Guan, Bryan Graham, Riccardo Bettati, and Wei Zhao. Using parasite flows to camouflage flow traffic. In *Proceedings of the 2002 IEEE Workshop on Information Assurance*, 2002.
- [88] Jana Dittmann, Danny Hesse, and Reyk Hillert. Steganography and steganalysis in voice-over ip scenarios: operational aspects and first experiences with a new steganalysis tool set. In Edward J. Delp and Ping Wah Wong, editors, *Security, Steganography, and Watermarking of Multimedia Contents*, volume 5681 of *Proceedings of SPIE*, pages 607–618. SPIE, 2005.
- [89] Yali Liu, Dipak Ghosal, Frederik Armknecht, Ahmad-Reza Sadeghi, Stefan Schulz, and Stefan Katzenbeisser. Hide and seek in time - robust covert timing channels. In Michael Backes and Peng Ning, editors, *ESORICS*, volume 5789 of *Lecture Notes in Computer Science*, pages 120–135. Springer, 2009.
- [90] Craig H. Rowland. Covert channel file transfer for linux. http://www-scf.usc.edu/~csci530l/downloads/covert_tcp.c, 2011.
- [91] Luis Campo Giralte, Cristina Conde, Isaac Martin de Diego, and Enrique Cabello. Incc: Hiding information by mimicking traffic in network flows. In

- 10th International Conference on Security and Cryptography (SECRYPT 2013)*, Reykjavik, Iceland, July 2013. SciTePress.
- [92] Shigang Chen and Qingguo Song. Perimeter-based defense against high bandwidth ddos attacks. *IEEE Transactions on Parallel and Distributed Systems*, 16:526–537, 2005.
- [93] Fariba Khan and Carl A. Gunter. Tiered incentives for integrity based queuing. In *Workshop on the Economics of Networks, Systems, and Computation (NetEcon'10)*, Vancouver, BC, Canada, October 2010. Usenix OSDI.
- [94] Luis Campo Giralte, Cristina Conde, Isaac Martin de Diego, and Enrique Cabello. Dlimiter : Mitigating distributed denial of service by using multiple discipline queues. In *International Conference on Mechatronics, Electronics and Automotive Engineering*, Cuernavaca, Mexico, November 2013. IEEE.
- [95] Zhang Shu and Partha Dasgupta. Denying denial-of-service attacks: A router based solution. In Hamid R. Arabnia and Youngsong Mun, editors, *International Conference on Internet Computing*, pages 301–307. CSREA Press, 2003.
- [96] Dan Jen, Michael Meisel, He Yan, Daniel Massey, Lan Wang, Beichuan Zhang, and Lixia Zhang. Towards a new internet routing architecture: Arguments for separating edges from transit core. In *Seventh ACM Workshop on Hot Topics in Networks (HotNets-VII)*, Calgary, Alberta, Canada, oct 2008.
- [97] Jarmo Mölsä. Effectiveness of rate-limiting in mitigating flooding dos attacks. In M. H. Hamza, editor, *Communications, Internet, and Information Technology*, pages 155–160. IASTED/ACTA Press, 2004.
- [98] Gal Badishi, Amir Herzberg, Idit Keidar, Oleg Romanov, and Avital Yachin. An empirical study of denial of service mitigation techniques. In *SRDS '08: Proceedings of the 2008 Symposium on Reliable Distributed Systems*, pages 115–124, Washington, DC, USA, 2008. IEEE Computer Society.
- [99] Haidar Safa, Mohamad Chouman, Hassan Artail, and Marcel Karam. A collaborative defense mechanism against syn flooding attacks in ip networks. *J. Netw. Comput. Appl.*, 31(4):509–534, 2008.
- [100] Wei-Zhou Lu, Wei-Xuan Gu, and Shun-Zheng Yu. One-way queuing delay measurement and its application on detecting ddos attack. *J. Netw. Comput. Appl.*, 32(2):367–376, 2009.
- [101] Supranamaya Ranjan, Ram Swaminathan, Mustafa Uysal, Antonio Nucci, and Edward W. Knightly. Ddos-shield: Ddos-resilient scheduling to

- counter application layer attacks. *IEEE/ACM Trans. Netw.*, 17(1):26–39, 2009.
- [102] Elaine Shi, Ion Stoica, David Andersen, and Adrian Perrig. OverDoSe: A generic DDoS protection service using an overlay network. Technical Report CMU-CS-06-114, Carnegie Mellon University Computer Science Department, February 2006.
- [103] XiaoFeng Wang and Michael K. Reiter. Mitigating bandwidth-exhaustion attacks using congestion puzzles. In *Proceedings of the 11th ACM conference on Computer and communications security, CCS '04*, pages 257–267, New York, NY, USA, 2004. ACM.
- [104] Nicholas A. Fraser, Douglas J. Kelly, Richard A. Raines, Rusty O. Baldwin, and Barry E. Mullins. Using client puzzles to mitigate distributed denial of service attacks in the tor anonymous routing environment. In *Proceedings of IEEE International Conference on Communications, ICC 2007, Glasgow, Scotland, 24-28 June 2007*, pages 1197–1202, 2007.
- [105] Markus Goldstein, Christoph H. Lampert, Matthias Reif, Armin Stahl, and Thomas M. Breuel. Bayes optimal ddos mitigation by adaptive history-based ip filtering. In *ICN*, pages 174–179. IEEE Computer Society, 2008.
- [106] Netfilter. <http://www.netfilter.org>, 2013.
- [107] Tor. <https://www.torproject.org/>, 2013.
- [108] Markus Goldstein, Matthias Reif, Armin Stahl, and Thomas Breuel. High performance traffic shaping for ddos mitigation. In *CONEXT '08: Proceedings of the 2008 ACM CoNEXT Conference*, pages 1–2, New York, NY, USA, 2008. ACM.
- [109] Hiroaki Hazeyama, Youki Kadobayashi, Daisuke Miyamoto, and Masafumi Oe. An autonomous architecture for inter-domain traceback across the borders of network operation. In *Proceedings of the 11th IEEE Symposium on Computers and Communications, ISCC '06*, pages 378–385, Washington, DC, USA, 2006. IEEE Computer Society.
- [110] Boosting japan’s cybersecurity.
<http://www.japantimes.co.jp/text/eo20121026a1.html>, 2012.
- [111] The pirate bay is latest site to be hit by ddos attack.
<http://www.computerweekly.com/news/2240150412/The-Pirate-Bay-is-latest-site-to-be-hit-by-a-DDoS-attack>, 2012.
- [112] Bank of america website slows after islamic hacker threats.
http://www.informationweek.com/security/attacks/bank-of-america-website-slows-after-islam/240007581?cid=edit_stub_WST, 2012.

- [113] The ddos that knocked spamhaus offline (and how we mitigated it). <http://blog.cloudflare.com/the-ddos-that-knocked-spamhaus-offline-and-ho>, 2013.
- [114] Hitesh Ballani and Paul Francis. Mitigating dns dos attacks. In *Proceedings of the 15th ACM conference on Computer and communications security*, CCS '08, pages 189–198, New York, NY, USA, 2008. ACM.
- [115] Armin Büscher and Thorsten Holz. Tracking ddos attacks: insights into the business of disrupting the web. In *Proceedings of the 5th USENIX conference on Large-Scale Exploits and Emergent Threats*, LEET'12, pages 8–8, Berkeley, CA, USA, 2012. USENIX Association.
- [116] Software dbus. <http://www.freedesktop.org/wiki/Software/dbus>, 2013.
- [117] IEEE Computer Society. Media access control (mac) bridges and virtual bridge local area networks. <http://standards.ieee.org/getieee802/download/802.1Q-2011.pdf>, 2012.
- [118] The tls protocol. <http://www.ietf.org/rfc/rfc2246.txt>, 2013.
- [119] Tcpdump. <http://www.tcpdump.org/>, 2013.
- [120] Xiapu Luo. Optimizing the pulsing denial-of-service attacks. In *Proceedings of the 2005 International Conference on Dependable Systems and Networks*, DSN '05, pages 582–591, Washington, DC, USA, 2005. IEEE Computer Society.
- [121] Xiapu Luo, Edmond W. W. Chan, and Rocky K. C. Chang. Detecting pulsing denial-of-service attacks with nondeterministic attack intervals. *EURASIP J. Adv. Signal Process*, 2009:8:1–8:13, January 2009.
- [122] Kuo Dong, Shoubao Yang, and Shaolin Wang. Analysis of low-rate tcp dos attack against fast tcp. *Intelligent Systems Design and Applications, International Conference on*, 3:86–91, 2006.
- [123] Luis Campo-Giralte, Ricardo Jimenez-Peris, and Marta Patino-Martinez. Polyvaccine: Protecting web servers against zero-day, polymorphic and metamorphic exploits. In *Proceedings of the 2009 28th IEEE International Symposium on Reliable Distributed Systems*, SRDS '09, pages 91–99, Washington, DC, USA, 2009. IEEE Computer Society.
- [124] Google's official googlebot. <http://www.google.com/bot.html>, 2013.
- [125] Acunetix web application security. <http://www.acunetix.com/>, 2013.
- [126] Portknocking. <http://www.portknocking.org>, 2013.
- [127] Rennie Degraaf, John Aycock, and Michael Jacobson. Improved port knocking with strong authentication. In *In Proc. 21st Annual Computer Security Applications Conference (ACSAC 2005)*, pages 409–418. Springer, 2005.

- [128] Muhammad Tariq, M. S. Baig, and M. Tariq Saeed. Associating the authentication and connection-establishment phases in passive authorization techniques, 2008.
- [129] The rc4-hmac kerberos encryption types used by microsoft windows. <http://www.ietf.org/rfc/rfc4757.txt>, 2013.
- [130] Snort. <http://www.snort.org/>, 2013.
- [131] Opendpi. <http://www.ipoque.com/en/products/pace>, 2013.
- [132] Luis Campo Giralte, Cristina Conde, Isaac Martin de Diego, and Enrique Cabello. Incc: Evading interception and inspection by mimicking traffic in network flows. In *International Joint Conference on e-business and Telecommunications*, Lecture Notes in Computer Science. Springer-Verlag, 2013.
- [133] Bittorrent. <http://bramcohen.com/>, 2013.
- [134] The bittorrent protocol specification, version 11031. http://bittorrent.org/beps/bep_0003.html, 2013.
- [135] Andreas Klein. Attacks on the rc4 stream cipher. *Des. Codes Cryptography*, 48(3):269–286, 2008.
- [136] Predicting and distinguishing attacks on rc4 keystream generator. In *EUROCRYPT*, pages 491–506, 2005.
- [137] Souradyuti Paul and Bart Preneel. A new weakness in the rc4 keystream generator and an approach to improve the security of the cipher. pages 245–259. 2004.
- [138] Monowar H. Bhuyan, D.K. Bhattacharyya, and J.K. Kalita. Surveying port scans and their detection methodologies. *Comput. J.*, 54(10):1565–1581, October 2011.
- [139] Hi-performance protocol identification engine. <http://sourceforge.net/projects/hippie/>, 2013.
- [140] Python. <http://www.python.org>, 2013.
- [141] Tcp extensions for high performance. <http://www.ietf.org/rfc/rfc1323.txt>, 2013.
- [142] Cliff Lampe and Erik Johnston. Follow the (slash) dot: effects of feedback on new members in an online community. In Mark Pendergast, Kjeld Schmidt, Gloria Mark, and Mark Ackerman, editors, *GROUP*, pages 11–20. ACM, 2005.
- [143] Daniel bernstein. <http://cr.yip.to/djb.html>, 2013.

- [144] Shunsuke Oshima, Takuo Nakashima, and Toshinori Sueyoshi. Ddos detection technique using statistical analysis to generate quick response time. *Broadband, Wireless Computing, Communication and Applications, International Conference on*, 0:672–677, 2010.
- [145] Daniel S. Yeung, Shuyuan Jin, and Xizhao Wang. Covariance-matrix modeling and detecting various flooding attacks. *IEEE Transactions on Systems, Man, and Cybernetics, Part A*, 37(2):157–169, 2007.
- [146] Linux programmer’s manual. http://www.kernel.org/doc/man-pages/online/pages/man3/pthread_setaffinity_np.3.html, 2012.
- [147] Gotocode. <http://www.gotocode.com>, 2010.
- [148] The gnu general public license. <http://www.gnu.org/licenses/gpl.html>, 2013.
- [149] Sip: Session initiation protocol. <http://www.ietf.org/rfc/rfc3261.txt>, 2013.