1. Introducción

1.1. Orígenes

A finales de Octubre de 2010, me puse en contacto con Pablo Nebrera con la intención de realizar mi Proyecto Fin de Carrera con él. Me propuso enfocarlo a los **Sistemas de Detección y Prevención de Intrusos (IDS/IPS)**. Estos sistemas son cada vez más necesarios en un mundo en el que la seguridad de la información se está convirtiendo en un activo muy importante dentro de toda institución, sea pública o privada, que gestione un volumen de datos importantes y sensibles de ser manipulados de forma maliciosa por terceras personas.

Tras una reunión con Pablo Nebrera y Jaime Nebrera, se concretaron los puntos y objetivos iniciales y se sentaron las bases de un Proyecto muy interesante y atractivo que estaría en estrecha colaboración con grupos de investigación de otras universidades. La finalidad del proyecto era mejorar algo en lo que ya se había trabajado mucho, se quería dar una vuelta de tuerca más y conseguir aumentar el rendimiento de una aplicación, llamada **Snort**, cuyo primer desarrollo se presentó en 1998 y que hoy en día está afianzada en el mundo de los IDS/IPS, contando con una amplia comunidad activa, con millones de descargas como aval y con cerca de 400.000 usuarios registrados[1].

1.2. Sistemas de Detección y Prevención de Intrusos (IDS/IPS): Snort

Hoy en día, en casi todos los entornos empresariales y públicos, desde una operadora de telefonía móvil hasta un organismo de índole público, se gestionan gran cantidad de datos importantes, como pueden ser los datos de los clientes en cartera, planes de proyectos futuros y demás contenido privado que debe ser protegido de intentos de robo o manipulación.

Cada vez hay más cantidad de datos disponibles en Internet, bien sea para ponerlos a disposición del usuario final de manera telemática, como el resumen de la vida laboral, o para realizar la gestión interna de una institución, como puede ser la comunicación e intercambio de datos entre dos sedes de una misma empresa.

Esta exposición de los datos a la red debería ir acompañada de un incremento de la seguridad en igual o mayor medida. No sería recomendable descuidar la seguridad, y por tanto aumentar la

1 www.snort.org

vulnerabilidad de la institución, sino que debería existir una conciencia del peligro que supone no usar los medios necesarios que prevengan este tipo de ataques que cada vez son más frecuentes y que son realizados por gente muy preparada.

Para cumplir con estos fines, Snort es una de las aplicaciones más utilizada de su categoría. **Snort es un proyecto de software libre** lanzado en 1998 y que pone a disposición su código fuente. Hoy es considerado como el estándar de facto entre los Sistemas de Detección y Prevención de Intrusos. Es capaz de analizar en tiempo real el tráfico de una red IP en busca de paquete sospechosos y enviar alertas a su administrador.

Al igual que sucede en el caso de cualquier antivirus, Snort toma decisiones en base a un conjunto de reglas que son leídas en el proceso de arranque de la aplicación. A continuación se muestra un ejemplo de una regla básica:

```
alert tcp any any -> 192.168.100.91 22 (msg:"trafico SSH"; sid:101;)
```

Añadiendo esta simple regla y ejecutando Snort, todo tráfico que tenga como IP y puerto de destino 192.168.100.91:22, siendo indiferente la IP y puerto de origen, generaría la siguiente alerta:

```
[**] [1:101:1] trafico SSH [**] [Priority: 0] {TCP} 192.168.101.106:58468
-> 192.168.100.91:22
```

Este es un ejemplo muy simple para tener una toma de contacto con el funcionamiento más básico de Snort, que servirá para comprender mejor los siguientes puntos de la introducción.

1.3. Motivación

A pesar de las bondades que aporta la implantación de un IDS/IPS, éste lleva incluido un coste computacional importante que si no se se solventa puede provocar la pérdida de paquetes durante el proceso de análisis del tráfico de la red y por tanto la posibilidad de que no se hayan detectado posibles ataques.

Se podría pensar que esto no deja de ser un problema menor puesto que con el paso de los años las prestaciones de los equipos informáticos irán en ascenso y por tanto el coste computacional no sería un factor crítico. No obstante, existen otros factores que indican lo contrario, como el incremento de la capacidad de las redes Ethernet o el aumento del número de reglas, que suele provocar una disminución en el rendimiento de cualquier IDS/IPS.

A continuación se mostrará una **estimación temporal del rendimiento** que podría sufrir una aplicación IDS. Esta estimación se fundamenta en hechos objetivos y podría aplicarse a cualquier IDS. Se tratará la evolución producida hasta ahora de la capacidad de las redes Ethernet, de las prestaciones de los equipos informáticos, del número de reglas usadas en Snort y de cómo ha afectado este incremento en el rendimiento de la aplicación. Y por último se realizará una estimación, en base a una extrapolación de estos parámetros, de cómo evolucionará el rendimiento final de un IDS en los próximos años.

1.3.1. Estimación del rendimiento en los próximos años

En este punto se realizará un estudio de los diversos factores que afectan en el rendimiento de un IDS. Este estudio se ha modelado matemáticamente como se explica a continuación.

El **rendimiento** (ŋ) de todo analizador de tráfico de red está supeditado a las prestaciones del equipo en el que se ejecuta, a la capacidad de tráfico de la red y al número de reglas, entre otros muchos factores. No obstante, estos tres factores podrían considerarse como los más determinantes a la hora de relacionar el rendimiento con determinadas condiciones del entorno. Es decir, la cuantificación del rendimiento podría obtenerse como la relación entre el **volumen de información analizada** en un segundo (Tput) y la **capacidad de la red** (C), con las **prestaciones del equipo** o sistema informático (P) y el **número de reglas** (N) como factores invariantes en el entorno. Mientras que el volumen de información analizada (Tput) se podrá calcular directamente en función de las prestaciones del equipo (P) y del número de reglas (N).

$$\eta = \frac{Tput}{C}$$

Por tanto, el concepto de rendimiento que se ha propuesto varía del siguiente modo:

- Aumenta proporcionalmente con las **prestaciones** del equipo (P)
- Disminuye a medida que aumenta el **número de reglas** (N). Más adelante se verá que esta relación influirá, en mayor o menor medida, dependiendo del algoritmo de búsqueda de patrones que se use, pudiendo incluso no influir en absoluto.
- Disminuye al aumentar la **capacidad** de la red (C)

Una vez aclarado el concepto propuesto de rendimiento de un analizador de tráfico, se mostrará una previsión de estos tres factores por separado, así como una estimación cualitativa de cómo afectarán estas previsiones en el rendimiento de los analizadores de tráfico. En adelante, en lugar de hacer referencia a los analizadores de tráfico en general, sólo se hará mención a Snort, por ser éste el analizador con el que se ha trabajado en el desarrollo del proyecto, pudiendo extender las siguientes consideraciones a cualquier otro analizador de tráfico existente.

1.3.1.1. Prestaciones del equipo

Para las próximas conclusiones se tendrá en consideración la **Ley de Moore**[2], relativa al aumento del número de transistores en un chip, en la que se establece un factor de crecimiento de 2x por cada dos años, lo que se traduce en un factor de 1.41x por año. Esta teoría, que data de 1965, ha predicho de manera muy fidedigna la evolución real de los procesadores hasta la actualidad, como se puede apreciar en la figura 1.1, donde se recogen, además de la evolución de los microprocesadores, la evolución de las memorias y la proyección desde 1975. Por tanto, se podría estimar, sin un margen de error elevado, que **el factor P aumentaría 1.41x por año**.

2 http://www.intel.com/about/companyinfo/museum/exhibits/moore.htm

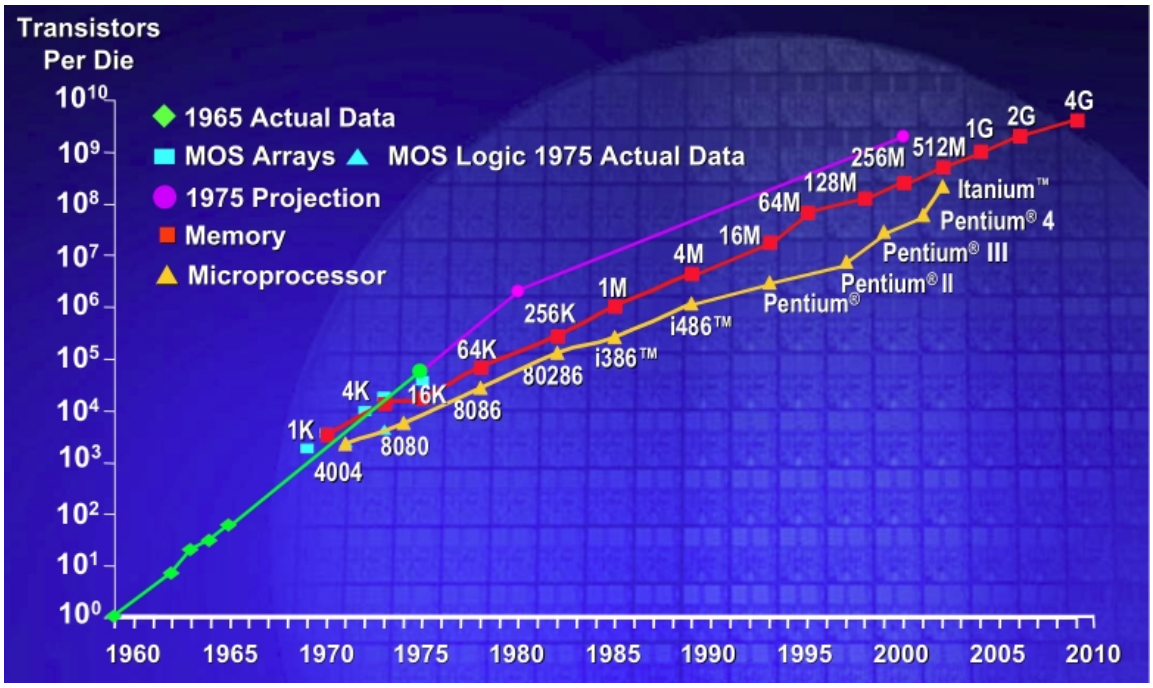


Figura 1.1: Ley de Moore para microprocesadores y chips de memoria. La curva morada es la proyección de la Ley de Moore basada en los datos recopilados hasta el año 1975. Notar la corrección existente alrededor de 1980. Fuente: Intel Corporation.

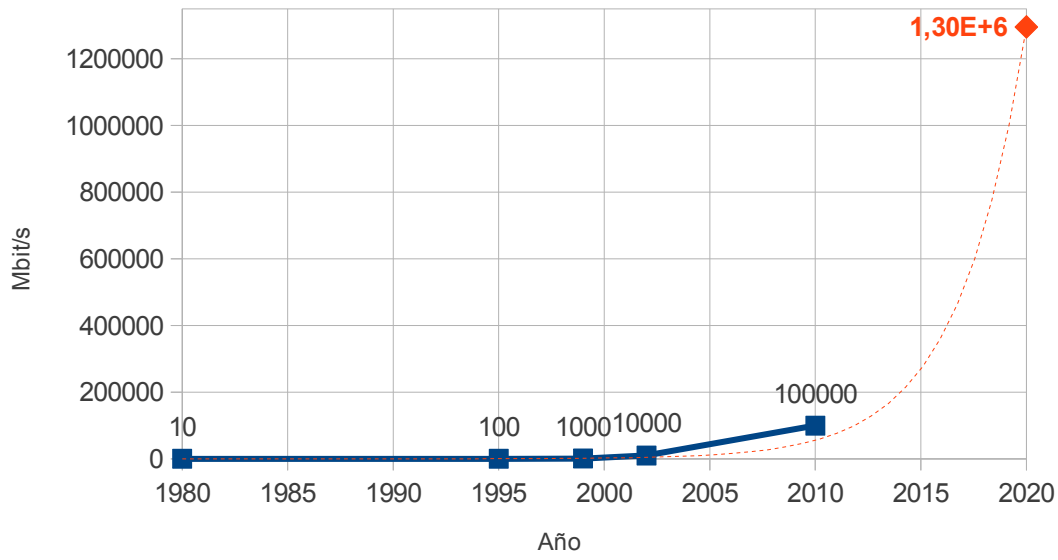
1.3.1.2. Capacidad de la red:

Desde el desarrollo de la tecnología Ethernet, la velocidad de transmisión de las redes de datos ha estado creciendo de manera exponencial con el paso de los años. Desde 1980, con la tecnología Ethernet, que alcanzaba velocidades de transmisión de datos de 10 Mbit/s, se ha ido evolucionando a pasos de gigante haciendo uso de las nuevas tecnologías de transmisión, como son la mejora del par trenzado, o más recientemente, el uso de tecnología óptica.

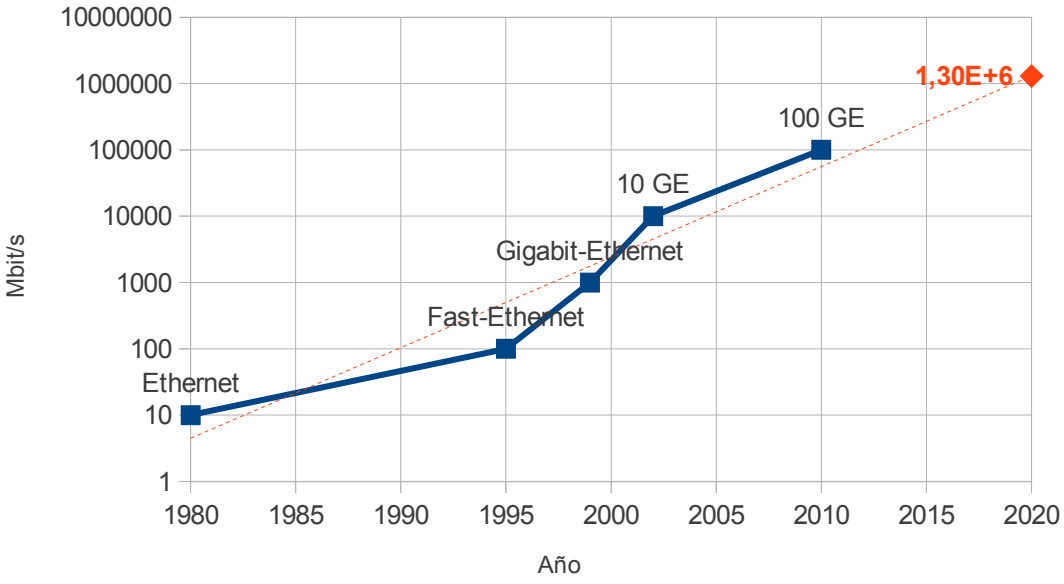
En las gráficas 1.1 y 1.2 se muestran la evolución anual de la capacidad de las redes Ethernet, en escala natural y semi-logarítmica, respectivamente, así como una estimación a largo plazo de la capacidad que podría llegar a alcanzarse en los próximos años.

Se puede apreciar que, durante el periodo de mayor evolución, desde 1995 hasta 2002, el **factor de crecimiento por año fue de 1.93x**, es decir, la velocidad de transmisión en este periodo se vio incrementada cada año en un factor de 1.93, bastante por encima del progreso previsto por la Ley de Moore para microprocesadores. Si extendemos este periodo, para tener una mejor perspectiva, y hacemos el cálculo desde 1995 hasta 2010, obtendremos un factor de crecimiento por año de 1.58x, también mayor que el factor previsto por la Ley de Moore.

Si se tiene en cuenta un enfoque conservador respecto a las previsiones de crecimiento, se podría estimar que **el factor C crecería 1.58x por año**.



Gráfica 1.1: Evolución anual de la capacidad de las redes Ethernet (escala natural) y estimación en el año 2020, basado en el cálculo de la curva de regresión exponencial entre 1980 y 2010.



Gráfica 1.2: Evolución anual de la capacidad de las redes Ethernet (escala semi-logarítmica) y estimación en el año 2020, basado en el cálculo de la curva de regresión exponencial entre 1980 y 2010.

1.3.1.3. Número de reglas

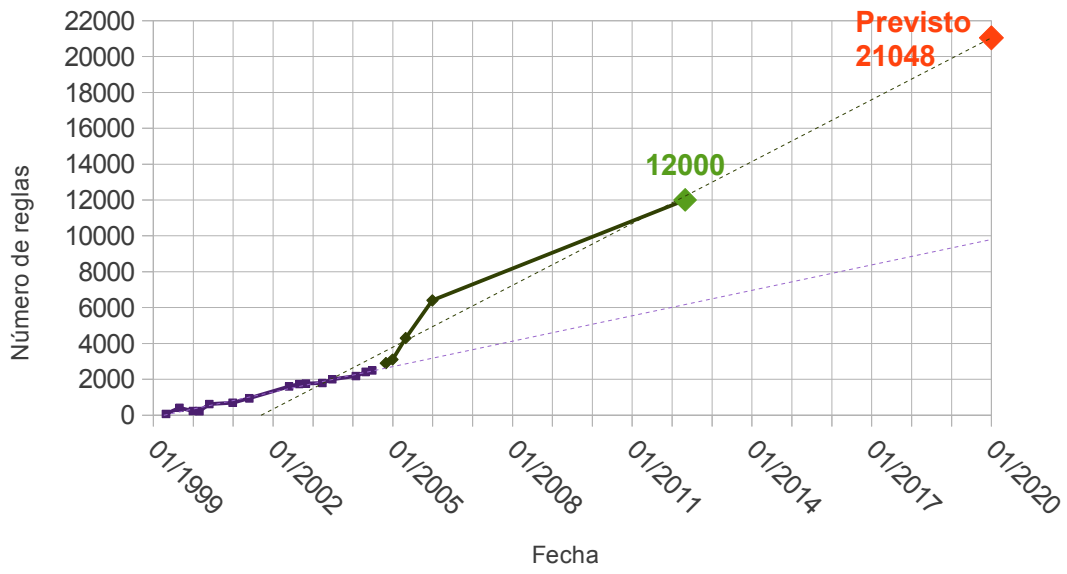
Tanto en un Sistema de Detección y Prevención de Intrusos, como en un antivirus, la decisión de

que una transmisión, o archivo, sea designado como un ataque, o un virus, dependerá de las reglas o firmas que la aplicación cargará al inicio de la ejecución.

Dichas reglas van evolucionando con el paso de los años, algunas se eliminarán o modificarán después de pasar por una revisión y otras muchas se irán incorporando.

Estas modificaciones provocan que el número de reglas vaya claramente en aumento. En la gráfica 1.3 se muestra una evolución del número de reglas desde 1999 hasta 2006. Se puede apreciar como sigue un crecimiento lineal hasta 2005, año en el que se produce un punto de inflexión tras el cual se incrementa considerablemente la pendiente hasta el año 2006. Para el estudio de la evolución del número de reglas sólo se han podido recopilar datos hasta 2006, pero se podría afirmar con bastante seguridad que se habrán producido más puntos de inflexión en el crecimiento a partir de entonces, debido, entre otros factores, a nuevas vulnerabilidades descubiertas. Por tanto, se podría obtener una aproximación bastante fiel del número de reglas que podrían alcanzarse en un futuro próximo.

En la misma gráfica se muestran también dos líneas de regresión, la primera calculada a partir de los datos proporcionados por el periodo comprendido entre 1999 y 2005, y la segunda escogiendo los datos del periodo que va desde 2005 hasta 2006, con la inclusión del dato de número de reglas actual. Al haberse dado un cambio de tendencia claro a partir del año 2005, se ha estimado el número futuro de reglas en base a una regresión lineal basada en el periodo desde 2005 hasta 2006 y añadiéndole el dato "aislado" del número de reglas actual. En este periodo obtendríamos que **el factor N crecería con una pendiente de 1150 reglas al año**. Téngase en cuenta que este factor no es exponencial, como lo eran los anteriores, sino lineal.



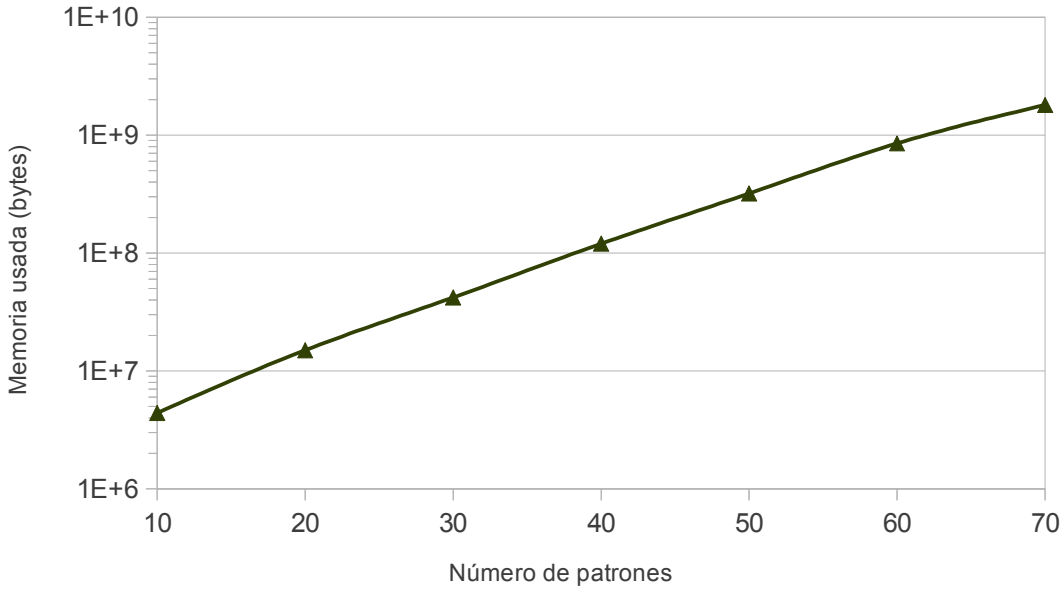
Gráfica 1.3: Evolución temporal del número de reglas. Recta de regresión evaluada desde 1999 hasta 2005. Recta de regresión evaluada desde 2005 hasta 2006 más el dato actual de número de reglas. Previsión de número de reglas en 2020 teniendo en cuenta la evolución desde 2005.

1.3.1.4. Relación entre el número de reglas y el rendimiento

A lo largo de la evolución de Snort se han ido incluyendo diversos algoritmos de búsqueda de patrones. En marzo de 1999[3] se implementó en el motor de búsqueda de patrones el algoritmo Boyer-Moore, rápido y simple, pero con la desventaja de no soportar búsqueda multi-patrón, esto quiere decir que para realizar una búsqueda de varios patrones se debía ejecutar el algoritmo una vez por cada patrón. Por tanto, cada aumento del número de reglas implicaba una merma en el rendimiento en la misma proporción.

Casi cuatro años más tarde, en octubre de 2002, se implementaba el algoritmo **Aho-Corasick**, un algoritmo de búsqueda multi-patrón que data de junio de 1975 y que sigue una estructura en árbol. A pesar de su antigüedad, sigue siendo un gran referente entre los algoritmos de búsqueda multi-patrón debido a su eficiencia en condiciones óptimas, ya que teóricamente no se ve afectado por el aumento del número de reglas, manteniendo un rendimiento constante. En capítulos posteriores se verá que esto no es así, puesto que en la práctica el alto consumo de memoria que requiere provoca una disminución en el rendimiento del algoritmo.

En la gráfica 1.4 se representa el número de patrones usados en la compilación de una estructura DFA (máquina de estados usada como estructura de detección por la implementación de Aho-Corasick en Snort) frente al consumo de memoria de dicha estructura. Se observa que el crecimiento del uso de memoria se produce de manera exponencial. Esta problemática es bien conocida y denominada como el "problema de la explosión de estados", que puede incurrir en una disminución del rendimiento de la búsqueda multi-patrón. En concreto, se puede obtener de la gráfica que **el factor de crecimiento del uso de memoria es de 1,105x por patrón**.



Gráfica 1.4: Relación, en escala semi-logarítmica, entre el número de patrones y la memoria usada (bytes) por una estructura DFA.

3fichero ChangeLog de Snort

En la figura 1.2 se muestra una gráfica en la que se puede apreciar la evolución lineal del consumo de memoria (en MB) frente al número de patrones, comparado con la disminución de la tasa de transferencia, o *throughput*, del proceso `grep` a medida que aumenta el número de patrones. En este caso, se puede observar como el crecimiento de la memoria usada no sigue una evolución exponencial, sino lineal; esto es así porque no todos los algoritmos de búsqueda de patrones ofrecen el mismo tipo de comportamiento en cuanto a incremento de uso de memoria se refiere. Aún así, lo interesante de esta figura es la comparación entre el consumo de memoria y el rendimiento en términos de tasa de transferencia. Esta figura ha sido obtenida de un artículo de Iulian Moraru, el cual se puede encontrar en la bibliografía. En él que se explica, entre otras cosas, cómo el rendimiento de un algoritmo de búsqueda de patrones se ve afectado por el consumo de memoria que éste requiere. A continuación se explicará qué relación existe entre el rendimiento de un proceso de búsqueda de patrones y la cantidad de memoria que consume.

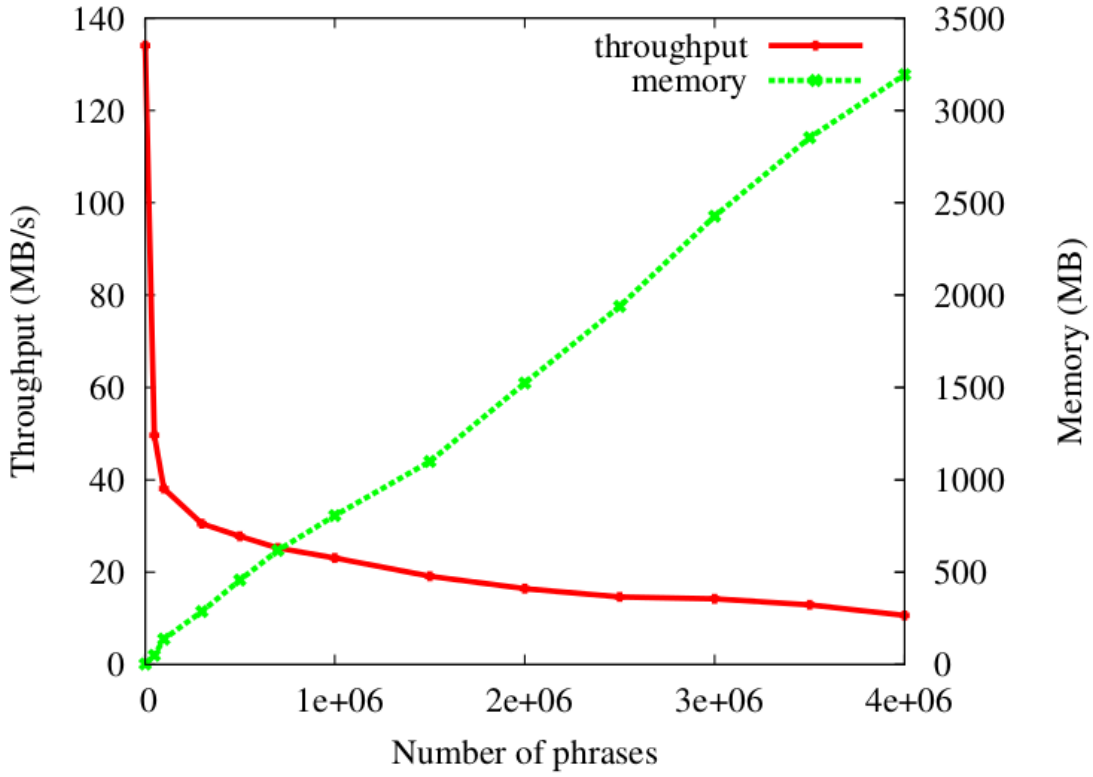


Figura 1.2: Tasa de transferencia (throughput) y consumo de memoria (memory) del proceso `grep` *frente al número de patrones usados. La longitud media de los patrones es de 29 caracteres. Fuente: Iulian Moraru and David G. Andersen. Fast Cache for Your Text: Accelerating Exact Pattern Matching with Feed-Forward Bloom Filters.*

La búsqueda de patrones se puede dividir en dos partes bien diferenciadas. La primera se trata de la creación de la estructura de búsqueda; esta estructura se construye en base a los patrones que se deseen buscar en un determinado texto. La segunda es el proceso de búsqueda de los patrones en un texto dado, en el que se hace uso de la estructura que se creó previamente.

Existen tres grupos principales de almacenamiento de datos: memoria caché, memoria RAM y disco duro, ordenadas de mayor a menor velocidad de acceso y de menor a mayor capacidad de

almacenamiento. Normalmente, en la ejecución de un algoritmo de este tipo no se requerirá el uso de memoria en disco duro, sino que debería bastar con la memoria caché y RAM. Parece lógico pensar que si la estructura de detección fuera lo suficientemente pequeña como para poder entrar en memoria caché, la ejecución del algoritmo sería más rápida que si tuviera que realizar accesos a memoria RAM. En este principio se basa el autor del artículo citado anteriormente.

En la gráfica de la figura 1.2 se aprecia una caída drástica de rendimiento cercana al origen del eje de abscisas. Según el autor, a partir de un cierto número, relativamente pequeño, de patrones la estructura de detección no es lo suficientemente pequeña como para poder entrar en su totalidad en memoria caché, por tanto se deben hacer más accesos a memoria RAM, y eso empeora el rendimiento.

De esta forma, se demuestra que a pesar de que teóricamente el algoritmo Aho-Corasick debería tener un rendimiento constante e independiente del número de patrones usados en la búsqueda, la realidad es bien distinta puesto que, como se ha visto en la gráfica de la figura 1.2, el número de reglas influirá en el consumo de memoria por parte de la estructura de detección y que, a su vez, al incrementar el uso de memoria, el rendimiento caerá drásticamente en torno a un cierto número pequeño de patrones, continuando la caída, pero a menor velocidad, a medida que el número de patrones aumenta.

Tanto la pendiente de la curva de caída, como el punto de inflexión que se produce en la curva del rendimiento, dependerán del algoritmo usado, con lo que deberá entenderse que las conclusiones finales mostradas a continuación podrán variar mucho en función del algoritmo que se quiera estudiar.

Con todo lo visto, se podría pensar que las conclusiones pueden llegar a ser algo subjetivas, en función de la interpretación personal que se tome de la evolución de los factores que hemos tratado. Sin embargo, para ser lo más objetivo posible, en las conclusiones se mostrará un enfoque con vistas a prever la mejor previsión futura posible del rendimiento de Snort, es decir, entre todos los márgenes que se han obtenido se ha escogido el límite que ofrecía mejores resultados en cuanto al rendimiento de la aplicación, siendo aún así estos resultados algo pesimistas. A continuación, en la tabla 1.1, se muestran los siguientes factores:

Factores	Prestaciones del equipo (P)	Capacidad de la red (C)	Número de reglas (N)	Memoria usada
Evolución	1,41x por año	1,58x por año	1150+ al año	1,105x por patrón

Tabla 1.1: Factores para el cálculo de la predicción del rendimiento de Snort

Hay que recordar que para el factor C se obtuvieron dos posibles resultados, en función del periodo de evolución de la red que se escogía, siendo 1,58x el resultado que más beneficiaba al rendimiento de Snort, en lugar de 1,93x.

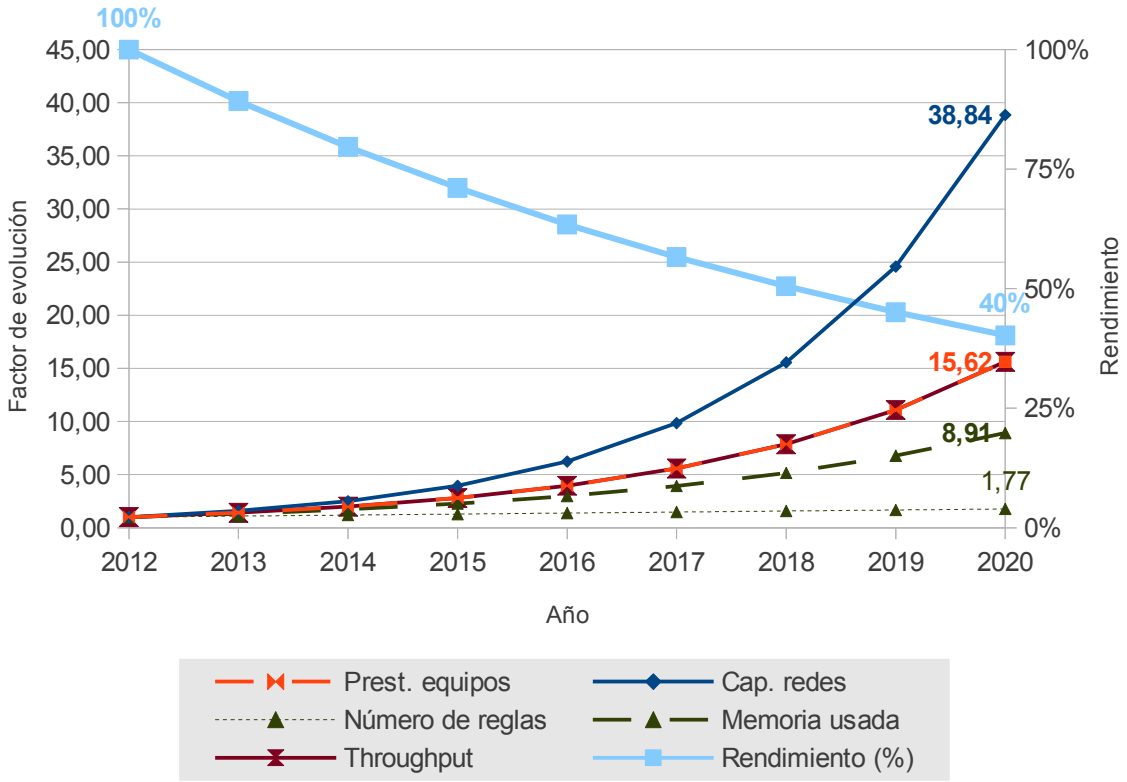
Otro aspecto a tener en cuenta es el valor del volumen de información analizada (o throughput). Para calcular este factor, se tendrá en cuenta que el crecimiento temporal de las prestaciones de los equipos puede darse a mayor o menor velocidad que el de la memoria usada. En el primer caso, cuando el incremento de las prestaciones sea más rápido que el del aumento de la cantidad de

memoria requerida, es decir que los equipos cuenten con más capacidad de memoria de la necesaria, el Throughput se podrá obtener directamente del factor P. Sin embargo, si la cantidad de memoria necesaria aumenta a mayor velocidad de lo que lo hacen las prestaciones, entonces el Tput debería ser calculado de manera más compleja, teniendo en cuenta un factor de corrección que dependa del aumento de memoria requerida.

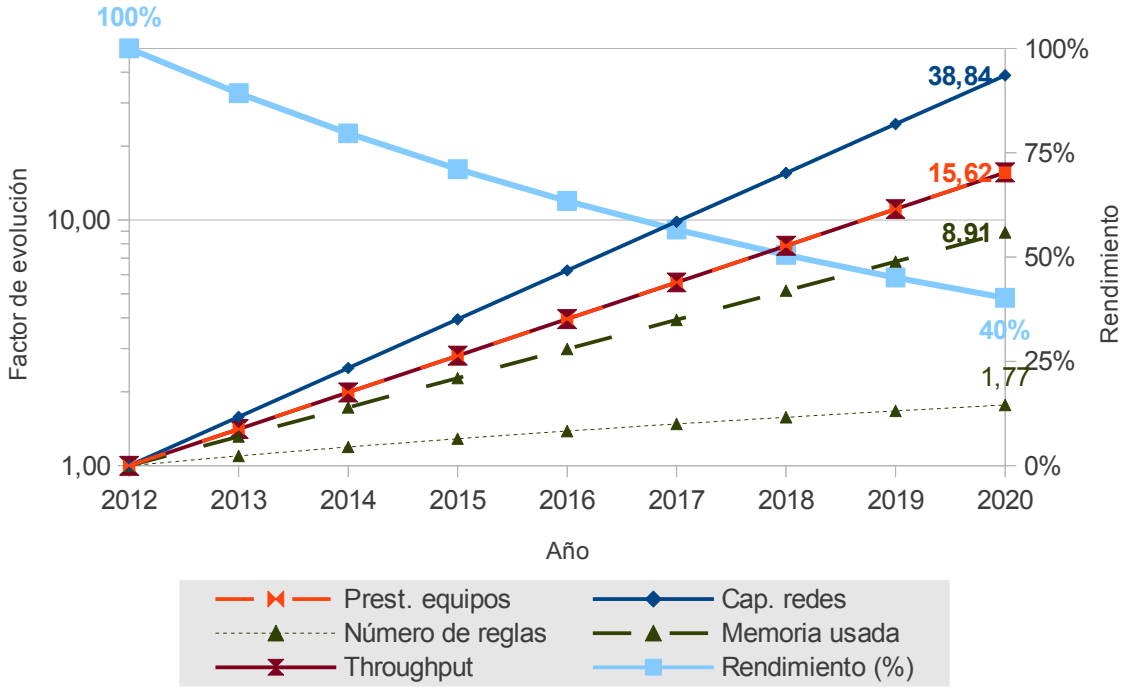
En nuestro caso, y una vez más, partiremos con un enfoque optimista, para así obtener una previsión de rendimiento mayor, y se descartará el hecho de que el uso de memoria sea un factor limitante, por lo que el throughput dependerá directa y únicamente de las prestaciones del equipo.

En las gráficas 1.5 y 1.6 se representa la evolución de los factores anteriores, siendo el año 2012 el punto de partida relativo y suponiendo que cada uno los factores comience con un valor unitario, que en el caso del rendimiento de Snort se traduce en un valor porcentual de 100. A partir de estos valores iniciales se muestra la evolución estimada que ha tenido cada uno de ellos con el paso de los años hasta alcanzar el año 2020.

El computo de la fórmula mostrada al inicio de este punto proporciona el valor de la previsión del rendimiento de Snort. Notar que en esta fórmula únicamente se se usan directamente el valor del rendimiento y el del throughput. No obstante, para hallar el throughput se utilizaban las prestaciones de los equipos y la memoria requerida por el IDS, que a su vez era calculada en función de la variación del número de reglas.



Gráfica 1.5: Estimación de la evolución anual del rendimiento de Snort (escala natural).



Gráfica 1.6: Estimación de la evolución anual del rendimiento de Snort (escala semi-logarítmica).

Para la representación de las gráficas 1.5 y 1.6 se han usado los resultados obtenidos al aplicar los factores de crecimiento. Estos resultados se muestran en la tabla 1.2.

En este estudio no se ha considerado un factor muy importante que tiene mucho peso en el futuro rendimiento de Snort. Este factor es la evolución que Snort consigue como aplicación con el paso de los años gracias a mejoras desarrolladas sobre el código fuente.

Año	Prestaciones del equipo (P)	Capacidad de la red (C)	Número de reglas (N)	Memoria usada	Throughput (Tput)	Rendimiento (%)
2012	1,00	1,00	1,00	1,00	1,00	100%
2013	1,41	1,58	1,10	1,31	1,41	89%
2014	1,99	2,50	1,19	1,73	1,99	80%
2015	2,80	3,94	1,29	2,27	2,80	71%
2016	3,95	6,23	1,38	2,98	3,95	63%
2017	5,57	9,85	1,48	3,92	5,57	57%
2018	7,86	15,56	1,58	5,16	7,86	51%
2019	11,08	24,58	1,67	6,78	11,08	45%
2020	15,62	38,84	1,77	8,91	15,62	40%

Tabla 1.2: Resultados de la evolución estimada anual del rendimiento de Snort.

El hecho de no haberlo incluido en la previsión no quiere decir que este factor no sea suficiente como para contrarrestar el previsible descenso del rendimiento de Snort, sino todo lo contrario. Desde sus inicios hasta la actualidad Snort ha recibido grandes aportes de conocimiento por parte de su comunidad, que ha contribuido activamente en el desarrollo de grandes mejoras. No obstante, ningún aporte es poco por pequeño que sea y desde nuestra posición se intentará, en la medida de lo posible, desarrollar una mejora de la aplicación o al menos dejar una puerta abierta para un futuro desarrollo.

1.4. Objetivos

Llegados a este punto del capítulo, el lector habrá entendido mejor cuál es el motivo por el que se quiere desarrollar una mejora para Snort. El siguiente paso será decidir **qué mejoras realizar** y qué motivos llevan a tomar tal determinación. Para ello se necesitará conocer qué puntos de la aplicación podrían ser a priori mejorables, así como entender el grado de dificultad al que se estaría sometido a la hora de llevar a cabo su desarrollo.

1.4.1. Puntos susceptibles de mejora

En los próximos puntos se describen las dos líneas de investigación que se han planteado:

- Un alto porcentaje del tráfico de una red, concretamente más del 90%, suele ser tráfico benigno, es decir, libre de ataques o contenido malicioso. Por tanto, a través del motor de búsqueda de patrones pasa una enorme cantidad de tráfico que termina por ser inofensivo. Si además se tiene en cuenta que aproximadamente el 30% del tiempo total que emplea Snort se dedica a la búsqueda de patrones, llegando a alcanzar el 80% para tráfico web intenso, se decidió que una buena opción sería la implementación de un **filtro previo a la búsqueda de patrones**, que descartara gran parte del tráfico benigno, para así evitar aliviar la carga computacional que lleva a cabo el motor de búsqueda de patrones.
- El algoritmo de búsqueda de patrones más eficiente que incluye Snort es Aho-Corasick que, como se ha dicho previamente, data de los años 70. A pesar de las continuas mejoras que se han realizado en la implementación del algoritmo dentro de Snort, no deja de ser un algoritmo ciertamente antiguo, aunque no por ello malo, prueba de ello es que sigue siendo un referente en la búsqueda de patrones. No obstante, se planteó la posibilidad de **aprovechar en mayor medida las capacidades que ofrecen los procesadores actuales**. En la actualidad existen librerías que aportan funciones optimizadas para la búsqueda de patrones, como es el caso de las librerías IPP de Intel. Son muchos los equipos que cuentan con la tecnología suficiente para implementar estas librerías, pero raramente son explotadas al máximo. Por tanto, otro camino de desarrollo podría ser el desarrollo de un algoritmo que aprovechara estas ventajas.

A continuación se detallará con más profundidad cada una de las propuestas planteadas.

1.4.2. Filtro previo: sigMatch

Jignesh M. Patel, profesor e investigador de la Universidad de Wisconsin-Madison, publicó en 2010 un estudio acerca de la implementación de un filtro para la búsqueda de patrones, bautizado como **sigMatch**. Parte del código desarrollado se liberó y fue publicado junto con el artículo. Dicho código liberado estaba escrito en Perl y C e implementaba una prueba de rendimiento sobre el antivirus ClamAV.

El objetivo de tal implementación es básicamente el que se ha explicado en el punto anterior: se cuenta con un conjunto de reglas o firmas y un texto en el que inspeccionar; previo a la búsqueda de patrones, se realiza un filtrado en el que se descartaría gran parte del texto y se dejaría pasar hacia el motor de búsqueda de patrones un porcentaje bastante menor formado por posibles ataques. Esta solución propuesta asegura que ningún ataque podrá ser omitido del análisis definitivo.

En la figura 1.3 se muestra un esquema del funcionamiento del proceso de búsqueda de patrones en Snort. El motor de búsqueda multi-patrón (MPSE, del inglés Multi-Pattern Search Engine) decide, basándose en un conjunto de firmas, si un paquete se considera benigno, o no. En el caso de que se trate de un paquete malicioso se enviaría la alerta correspondiente.

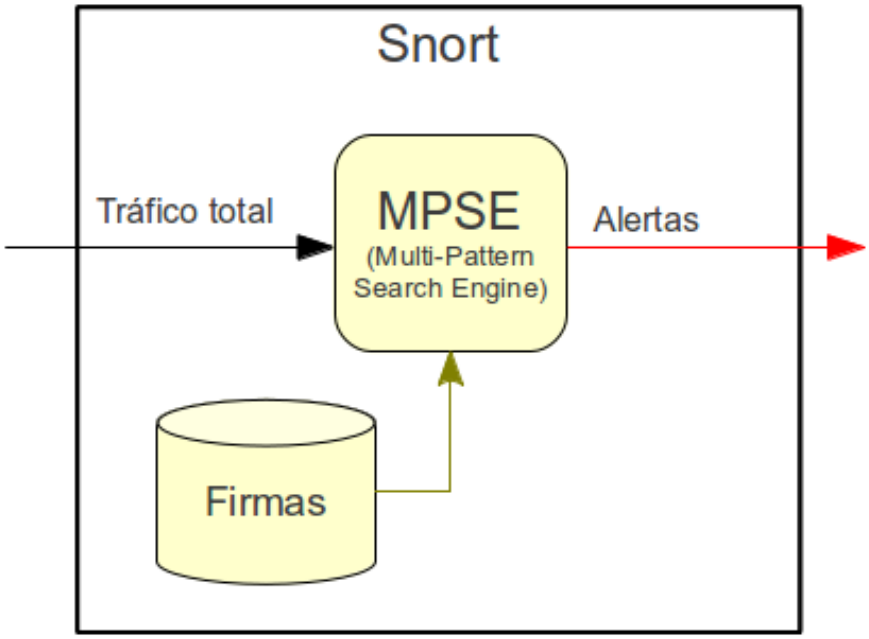


Figura 1.3: Proceso de búsqueda de patrones en Snort.

Para llegar a esta decisión, uno de los pasos que realiza el MPSE es llamar al algoritmo de búsqueda de patrones. Este algoritmo se especifica en el archivo de configuración de Snort (`snort.conf`). Actualmente, teniendo en cuenta el número de reglas existentes, el algoritmo de búsqueda que está ofreciendo mejores resultados es Aho-Corasick, y por tanto será el que se utilizará a lo largo del proyecto.

Dada la estructura del motor de búsqueda de patrones de Snort mostrada en la figura 1.3, se ha propuesto la implementación de sigMatch en Snort que se muestra en la figura 1.4.

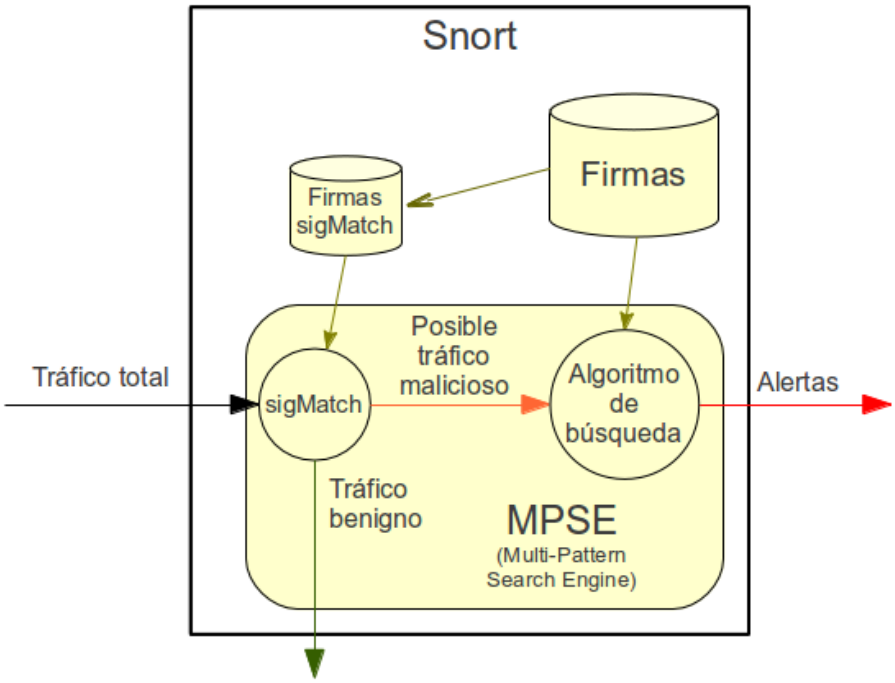


Figura 1.4: sigMatch como filtro previo al MPSE de Snort.

Una vez que llega un paquete al MPSE, este no lo redirige al algoritmo de búsqueda directamente, sino que lo hace pasar antes por el filtro previo sigMatch, que decidirá si es un posible ataque o, si por el contrario, es con total seguridad tráfico benigno.

Para tomar tal decisión, sigMatch se basa en las firmas que posteriormente usará el algoritmo de búsqueda, realiza un estudio de ellas, en las que obtiene ciertos patrones comunes, y posteriormente procede a realizar la búsqueda en cada paquete que entra al MPSE.

En el capítulo tres se muestra en profundidad el funcionamiento de sigMatch.

1.4.3. Uso y aprovechamiento de la tecnología actual

Intel, de un cierto tiempo hasta hoy, está trabajando en procesadores de altas prestaciones que incluyen un conjunto de instrucciones SIMD (Single Instruction, Multiple Data) denominadas SSE (Streaming SIMD Extensions). Existen librerías de Intel, conocidas como **IPP** (Integrated Performance Primitives), que cuentan con funciones que aprovechan estas instrucciones para la búsqueda de patrones.

Se podría pensar que el uso de estas funciones mejoraría el rendimiento de la búsqueda de patrones

en Snort. Sin embargo, hasta hoy, Snort no hace uso de ellas. Por tanto, otro camino de desarrollo que se propuso fue la inclusión del uso de librerías de Intel, bien fuera mejorando el filtro previo a la búsqueda de patrones o, directamente, creando un propio algoritmo que sustituyera a Aho-Corasick.

1.5. Fases del proyecto

A continuación, se detalla la evolución temporal que ha tenido el proyecto, desde sus inicios en Noviembre de 2011 hasta que se dio por finalizado en Mayo de 2012. Se incluye un diagrama de Gantt en la figura 1.5 para que sirva de ayuda.

I. sigMatch: En esta fase se procedió al estudio del funcionamiento de sigMatch: cómo se recopilan las reglas "resumidas" a partir de un conjunto de reglas dadas, cómo se crea la estructura de detección, etc. Aunque el código aportado por los autores sirvió de ayuda para dar los primeros pasos, éste no funcionaba correctamente y se empleó parte del tiempo en corregirlo, reescribiendo gran parte del código. No obstante, esta fase de corrección sirvió de ayuda para entender mejor su funcionamiento.

II. Snort: Es una aplicación escrita en C y que cuenta con muchos aportes de la comunidad. El código fuente principal, obviando preprocesadores y plugins, ocupa unos 3MB, lo que se traduce en más de 100.000 líneas de código. Por tanto, se podría decir que una fase importante del proyecto ha sido el estudio del funcionamiento interno de Snort: cómo se estructura el código, cómo funciona la parte de la búsqueda de patrones, dónde se decide cuándo se debe lanzar una alerta, etc. Todo este estudio fue necesario para decidir donde se

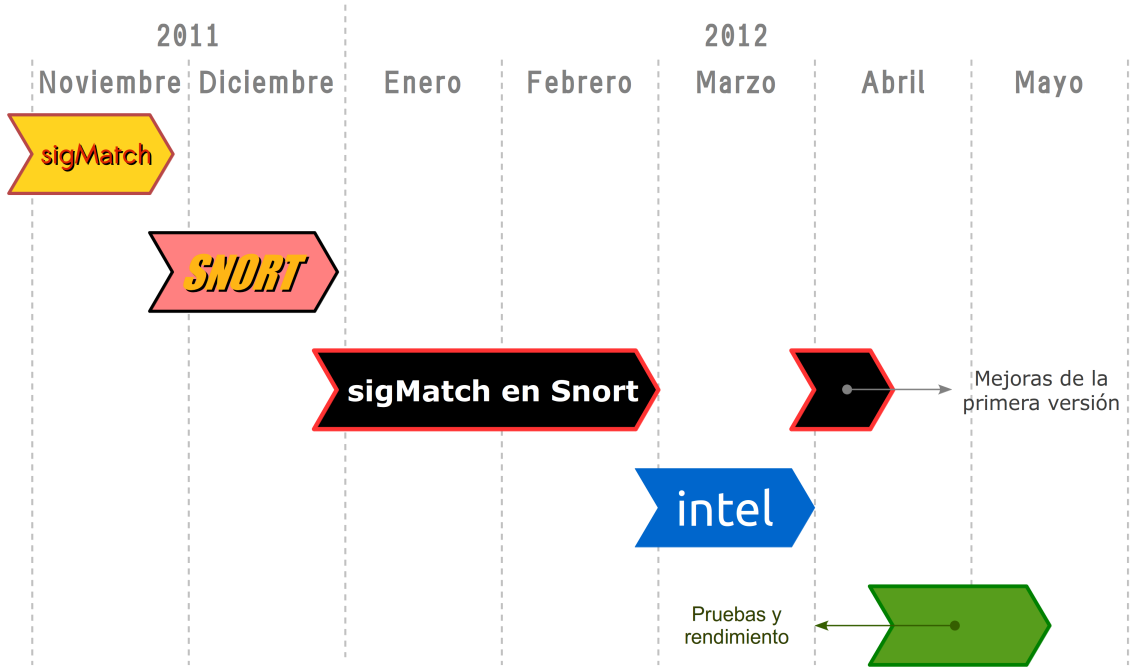


Figura 1.5: Fases del proyecto.

debía incluir el filtro previo al algoritmo de búsqueda de patrones.

III. sigMatch en Snort: Una vez tomada la decisión, se inició la fase de implementación del filtro en Snort. Esta fase contó en primer lugar con la creación del conjunto de firmas resumidas, en base a las firmas que usaba el algoritmo de búsqueda de patrones, seguida de la creación de la estructura de detección, propuesta por Patel y su equipo, y por último con la creación del método de detección. En segundo lugar, y una vez que se verificó el correcto funcionamiento, se realizaron pruebas del rendimiento obtenido. Por último, se continuó con el desarrollo de nuevas mejoras en la estructura de detección de sigMatch.

IV. Intel: En este punto del proyecto se inicia la fase de aprovechamiento de las capacidades que ofrecen los procesadores actuales. Para ello se diseñó un algoritmo que pudiera aprovechar mejor las prestaciones de los procesadores Intel, haciendo uso de las librerías IPP de Intel que incluyen un gran abanico de funciones optimizadas.

V. Mejoras de la primera versión: Se retoma el estudio de la primera versión del filtro, incorporando nuevas funcionalidades e ideas, adquiridas con el estudio de la algoritmia contemporánea, que podrían mejorar el rendimiento.

VI. Pruebas y conclusiones: Una vez finalizadas las partes de las que constaba el proyecto, se recopilan las pruebas de rendimiento realizadas y se muestran unas conclusiones acerca de todo el trabajo realizado, proponiendo una línea futura de desarrollo en base a la experiencia adquirida.