Universitat Oberta de Catalunya

uoc.edu

Detección de amenazas de seguridad de red a través de canales TLS

Luis Guillén Civera
Máster en Seguridad de las Tecnologías de la Información y de las Comunicaciones
Ad-hoc

Jordi Serra Ruiz

11/06/2018

Índice

Lista de figuras

Licencia

Ficha del Trabajo Final

Título del trabajo:	Detección de amenazas de seguridad de red a través de canales TLS
Nombre del autor:	Luis Guillén Civera
Nombre del consultor:	Jordi Serra Ruiz
Nombre del PRA:	
Fecha de entrega (mm/aaaa):	06/2018
Titulación:	Máster en Seguridad de las Tecnologías de la Información y de las Comunicaciones
Área del trabajo:	Ad-hoc
Idioma del trabajo:	Castellano
Palabras clave:	ids tls machine learning

Resumen del Trabajo (máximo 250 palabras):

La capa de seguridad TLS proporciona servicios de seguridad a protocolos como HTTP. Su éxito es indiscutible ya que en el último año más del 50% del tráfico de internet usaba HTTPS. Esto sin duda es una buena noticia porque quiere decir que Internet es más segura para los usuarios.

Sin embargo, los creadores de malware también se están adaptado a este nuevo escenario y, según estudios recientes, más del 20% del malware utiliza el protocolo TLS. Esto es un problema porque los detectores tradicionales de malware dejan de funcionan.

En la actualidad existen soluciones que consisten en romper la privacidad de los usuarios introduciendo elementos intermedios que generan las conexiones en lugar de los usuarios para así poder inspeccionar el tráfico en texto plano. Sin embargo, estas soluciones no están exentas de problemas.

Por ello surge la necesidad de otro tipo de estrategias para abordar el problema de la detección de malware. Una de ellas es el uso de algoritmos de aprendizaje automático que, en base a un conjunto de entrenamiento de conexiones cifradas, creen modelos capaces de detectar el malware sin necesidad de romper la privacidad de los usuarios.

En este proyecto se estudia el uso de algoritmos de aprendizaje automático aplicado a conexiones HTTPS. Para ello se utilizarán conjuntos de datos públicos y se crearán modelos que traten de dar solución al problema. Finalmente se desarrollará un software llamado luIDS que usará los modelos creados anteriormente para detectar malware en "tiempo real".

Abstract (in English, 200 words or less):

The TLS security layer provides security services to protocols such as HTTP. Its success is unquestionable since in the last year more than 50% of the internet traffic used HTTPS. It is certainly good news because it means that the internet is safer than ever for users.

However, malware developers are also adapting to this new scenario and, according to recent studies, more than 20% of malware uses the TLS protocol. This is a problem because traditional malware detectors are not working now.

Currently there are solutions that break the privacy of users by placing elements between users and servers in order to inspect traffic in plain text. However, these solutions have problems.

Therefore, other types of strategies are needed to solve the problem of malware detection. One of them is the use of machine learning algorithms that, based on a training set of encrypted connections, creates models capable of detecting malware without breaking the privacy of users.

In this project the use of machine learning algorithms applied to HTTPS connections is studied. To achieve this, public data sets will be used and models will be created to try to solve the problem. Finally, a software called luIDS will be developed and it will use the previously created models to detect malware in "real time".

Chapter 1. Introducción

Este capítulo pretende ser una introducción al proyecto. A lo largo del mismo se determinará el contexto en el que se engloba, se fijarán los objetivos, la metodología y la planificación. Finalmente se enumerarán los productos adjuntos a la presente memoria y una breve descripción del contenido de los siguientes capítulos.

1.1. Contexto y justificación del Trabajo

Según este artículo [1], en Agosto de 2015 tan sólo el 2,21% de las muestras de *malware* empleaban el protocolo *TLS* para cifrar su conexión y evadir así a los sistemas de detección. En mayo de 2017 la cifra se incrementó hasta el 21,44%. Durante el mismo periodo de tiempo, de sólo 0,12% del malware que usaba TLS de manera exclusiva (que no usaba al mismo tiempo conexiones sin cifrar), dicha cantidad se ha incrementado al 4,45%. Otros fabricantes de soluciones de ciberseguridad como Zscaler [2], afirman en [3] que cerca del 60% del tráfico que analizan se encuentra cifrado mediante TLS y que el número de muestras maliciosas que usan este protocolo se había duplicado tan sólo en 6 meses. Si a esta tendencia le sumamos la irrupción de entidades de certificación gratuitas como Let's Encrypt [4], todo hace suponer que el crecimiento y la adaptación continua del malware a este protocolo va a ir en aumento.

Un mecanismo de detección "tradicional" ha sido la utilización de técnicas tipo *Man-in-the-middle* (MITM), por parte de las soluciones de seguridad para así poder analizar en claro el tráfico y buscar posibles patrones. Sin embargo, esta técnica además de ser costosa computacionalmente requiere comprometer la privacidad de los usuarios. Por si esto fuera poco, algunas herramientas y sitios web no funcionan correctamente si se hace uso de estas técnicas ya que comprueban la validez de los certificados de forma independiente a las *Entidades de Certificación* (CA) presentes en el navegador o el sistema o si se hace uso de certificados del lado del cliente.

Por todo ello, se hace necesario el estudio y el desarrollo de nuevas herramientas que se adapten a este nuevo escenario. Y esto es justo lo que vamos a hacer en este proyecto: construir una herramienta que, haciendo uso principalmente de algoritmos de *Machine Learning* (ML), sea capaz de detectar conexiones cifradas mediante el protocolo TLS que provengan de malware.

1.2. Objetivos del trabajo

Los objetivos fueron marcados al inicio del proyecto y puestos por escrito en la primera *PEC*, así que este apartado será una transcripción de dichos objetivos sin alterar prácticamente ni una coma. De esta forma podremos evaluar de forma más objetiva su grado del cumplimiento en el capítulo Conclusiones y futuro.

Me fue imposible contactar con la persona que tenía asignada como tutor de empresa. Por ello, y ante la necesidad de sacar adelante el proyecto, yo mismo definí los objetivos en base al conocimiento adquirido durante el estudio del Estado del arte.

1.2.1. Enunciado

El enunciado del objetivo trata de ser SMART, esto es: específico, medible, alcanzable, realista y acotado en el tiempo. Dicho enunciado es el siguiente:

> Tener listo para el día 4 de Junio un modelo de Machine Learning que, en base a posibles anomalías en el tráfico TLS, sea capaz de distinguir la presencia de malware con una precisión de al menos el 95% y una tasa de falsos positivos de máximo el 3%. Junto al modelo se proporcionará una implementación a modo de prueba de concepto de dicho modelo que, en conjunción con otras herramientas, sea capaz de analizar en tiempo real y alerte de las posibles conexiones de malware. Además se proporcionará una memoria que detallará todo el proceso de desarrollo del proyecto.

La fecha indicada encajaba en el marco temporal definido por la UOC y las métricas establecidas las hice en base a resultados observados en trabajos similares durante el estudio del Estado del arte. También propuse la creación de una prueba de concepto para desmarcarme y hacer algo más original que lo que se había hecho en trabajos similares.

El modelo que realizaremos, será supervisado, lo que implica que necesitaremos de un *dataset* con muestras de tráfico correctamente etiquetadas para poder entrenarlo. Este aspecto es crítico y supone un riesgo claro para el proyecto, ya que todavía no tengo respuesta acerca de cómo voy a obtener estos dataset [1: Finalmente no obtuve ningún tipo de respuesta.]. Aun así defino una serie de alternativas que describo en el apartado riesgos [2: Se definió un apartado riesgos en el que enumeraba algunos de los riesgos del proyecto.]. La clasificación mínima aceptable sería una clasificación binaria de malware/no malware, sin embargo, sería conveniente disponer de mayor información para implementar también un multiclasificador.

A partir del dataset que se proporcione, habrá que realizar las "labores de limpieza" oportunas y se tomarán las decisiones en lo que respecta al número de datos de entrenamiento y de tests. En el problema concreto que nos ocupa, en una conexión TLS tenemos la siguiente información:

- Información de la conexión: ips, puertos
- Información transmitida en claro: negociación del cifrado elegido, certificados y claves
- Información del flujo de datos: bytes transmitidos, paquetes, tiempo entre paquetes, etc

A partir de dicha información, podemos definir dos líneas de estudio sobre las que construiremos nuestro modelo y que darán lugar a dos de los hitos que se definirán en la planificación:

- **Análisis del handshake**: aquí estudiaremos la información del *handshake* y trataremos de extraer caracteristicas. En este apartado pueden surgir también diversas ideas de optimización para la detección complementarias al modelo.
- **Análisis del flujo**: este aspecto es más complejo y hay más opciones para el estudio. En este caso estudiaremos la información que puede obtenerse con algunos protocolos como *Netflow* para extraer características. También veremos cómo obtener información de "contexto" que

pueda ser relevante (conexiones anteriores desde la misma ip, etc). Finalmente existe otra opción para el estudio que consiste en modelar el comportamiento. Esta última opción es más avanzada y compleja, por lo que solo se abordará si el tiempo lo permite.

En ambos casos, seleccionaremos 3 de los algoritmos de ML que consideremos más apropiados y los aplicaremos para entrenar nuestro modelo. Utilizando diversos métodos como el el *cross validation* mediremos la eficacia y generaremos una comparativa entre los modelos según tiempos de cómputo, precisión y falsos positivos. Además se iterará probando el comportamiento según parámetros de los algoritmos y la introducción de características.

Una vez que tengamos un modelo de clasificación, implementaremos una solución que funcione en tiempo real que proporcione predicción en el momento del handshake y tras el cierre de la conexión.

La solución implementada sería deseable que se integrase con Snort o con alguna otra solución IDS y que usase un modelo completo con todas las características. Además sería deseable que la solución dispusiese también de optimizaciones adicionales al modelo (comprobación certificados, dns, listas negras, etc).

1.2.2. Listado de objetivos y priorización

A continuación describo 3 categorías de objetivos. La primera son los objetivos mínimos que se han de alcanzar, la segunda son objetivos que se cumplirán si se cumple a la perfección con la planificación y la tercera son objetivos que se encuentran fuera de la planificación pero que se podrían realizar si llegase a sobrar tiempo.

Mínimos

- Crear modelo ML que clasifique malware/no malware según el handshake
- Crear modelo ML que clasifique malware/no malware según netflow
- Desarrollar solución básica IDS que implemente los modelos de forma independiente
- Desarrollo de la memoria

Deseable

- Crear modelo ML que sea multiclasificador según tipo de malware según el handshake
- Crear modelo ML que sea multiclasificador según tipo de malware según netflow
- Utilizar características de contexto para el clasificador de flujo
- La solución IDS emplea un modelo con todas las características (hanshake+flujo+ contexto)

Muy optimistas

- Integrar solución con IDS Snort o similar
- Se enumeran a nivel de concepto posibles optimizaciones del IDS ajenas al modelo ML específicas de TLS
- En la solución IDS se implementan las optimizaciones citadas

- Agregar al modelo de flujo características de comportamiento de la conexión como Stratosphere IPS

1.3. Enfoque y método seguido

A continuación se evaluarán las posibles estrategias para poder llevar a cabo el trabajo y se definirá la metodología utilizada.

1.3.1. Enfoque

Como se describió en el apartado anterior, el objetivo fundamental es construir una herramienta (aunque sea a modo de prueba de concepto) capaz de clasificar conexiones provenientes de malware en "tiempo real". Aquí hay varias opciones, que básicamente se pueden resumir en dos:

1. **Utilizar los logs de herramientas ya existentes** que analizan y procesan el tráfico TLS para posteriormente analizarlos y realizar la clasificación: esta opción es mucho más simple pero también es mucho menos potente ya que no tenemos prácticamente control y estamos muy limitados a las opciones que ofrezcan dichas herramientas. Además de esto existen dificultades de tipo técnico para asociar la información proveniente de diferentes herramientas y se pone en riesgo la construcción de una herramienta que cumpla con el requisito de "tiempo real".
2. **Crear un herramienta propia especializada en el tráfico TLS**, que implemente dicho análisis, proceso y posterior clasificación: esta opción es mucho más compleja técnicamente pero a su vez mucho más potente ya que tenemos el control total del proceso.

Tras analizar pros y contras de las opciones se optó por la segunda opción.

1.3.2. Metodología

La metodología seleccionada para desarrollar este proyecto es la popular metodología CRISP-DM [5]. Al margen de su popularidad y de ser una metodología especializada en proyectos de minería de datos, lo que más me ha atraído y el principal factor por el que he elegido esta metodología es su naturaleza iterativa.

En dicha metodología hay dos aspectos clave: la adopción de una estrategia de calidad total con un ciclo PDCA ("Plan", "Do", "Check", "Act") y la visión de un proyecto de minería de datos como una secuencia de fases que describiré posteriormente.

Dentro del ciclo PDCA, el "check" corresponde con las revisiones definidas en cada hito de la planificación (se verá posteriormente) y derivado del proceso de revisión se toman una serie de medidas correctivas y la replanificación.

En lo que respecta a la secuencia de fases, la secuencia establecida en la metodología puede verse en el siguiente diagrama:

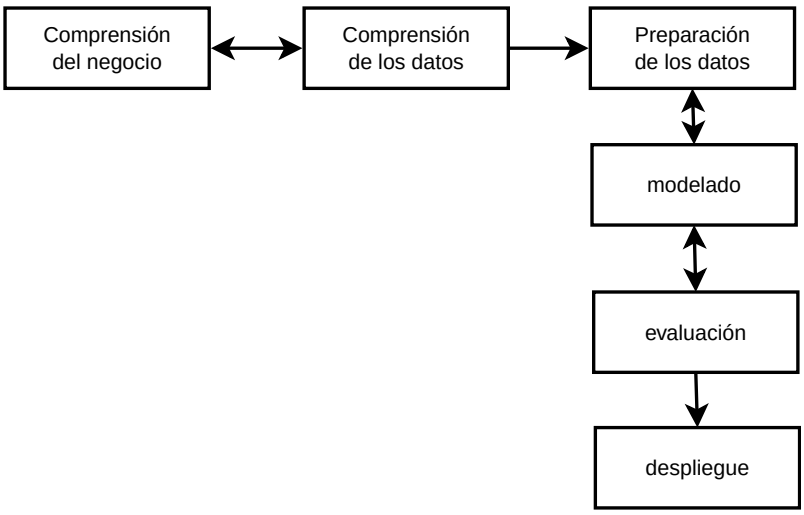

Figura 1. Diagrama de fases de la metodología CRISP-DM

Existe una descripción de la metodología en el magnífico libro "Minería de datos. Modelos y algoritmos" [6].

1.4. Planificación del Trabajo

Junto con los objetivos enumerados en el apartado Objetivos del trabajo, la planificación se detalló en la primera de las PEC y se realizaron cambios a la misma en las consecutivas PEC, tomando decisiones y medidas correctoras para asegurar con el cumplimiento de los objetivos principales proyecto. De la misma forma que se hizo con los objetivos, dejo a continuación una transcripción de la planificación inicial para así posteriormente evaluarla de manera objetiva en el apartado Conclusiones y futuro.

1.4.1. Definición de los hitos

La planificación está diseñada de acuerdo al marco temporal marcado por la UOC de entrega de PECs. Hay establecidos 3 hitos con un margen de 4 semanas entre cada hito, lo que hace un total de 12 semanas para realizar el proyecto.

- Hito 1: Clasificador ML con información del handshake (PEC 2)
- Hito 2: Clasificador ML con información del flujo (PEC 3)
- Hito 3: Implementación de solución NIDS y entrega de memoria (PEC 4)
- Hito 4: Presentación y vídeo (PEC 5)
- Hito 5: Fin de proyecto

La disponibilidad es de 5 días a la semana con 5 horas cada día, lo que hace un total de 12 x 5 x 5 = 300 horas, que son las horas que hay que dedicar al proyecto.

En el desglose de días de la planificación incluida en la primera PEC se tuvieron en cuenta los días festivos. No lo incluyo por no ser relevante para este documento.

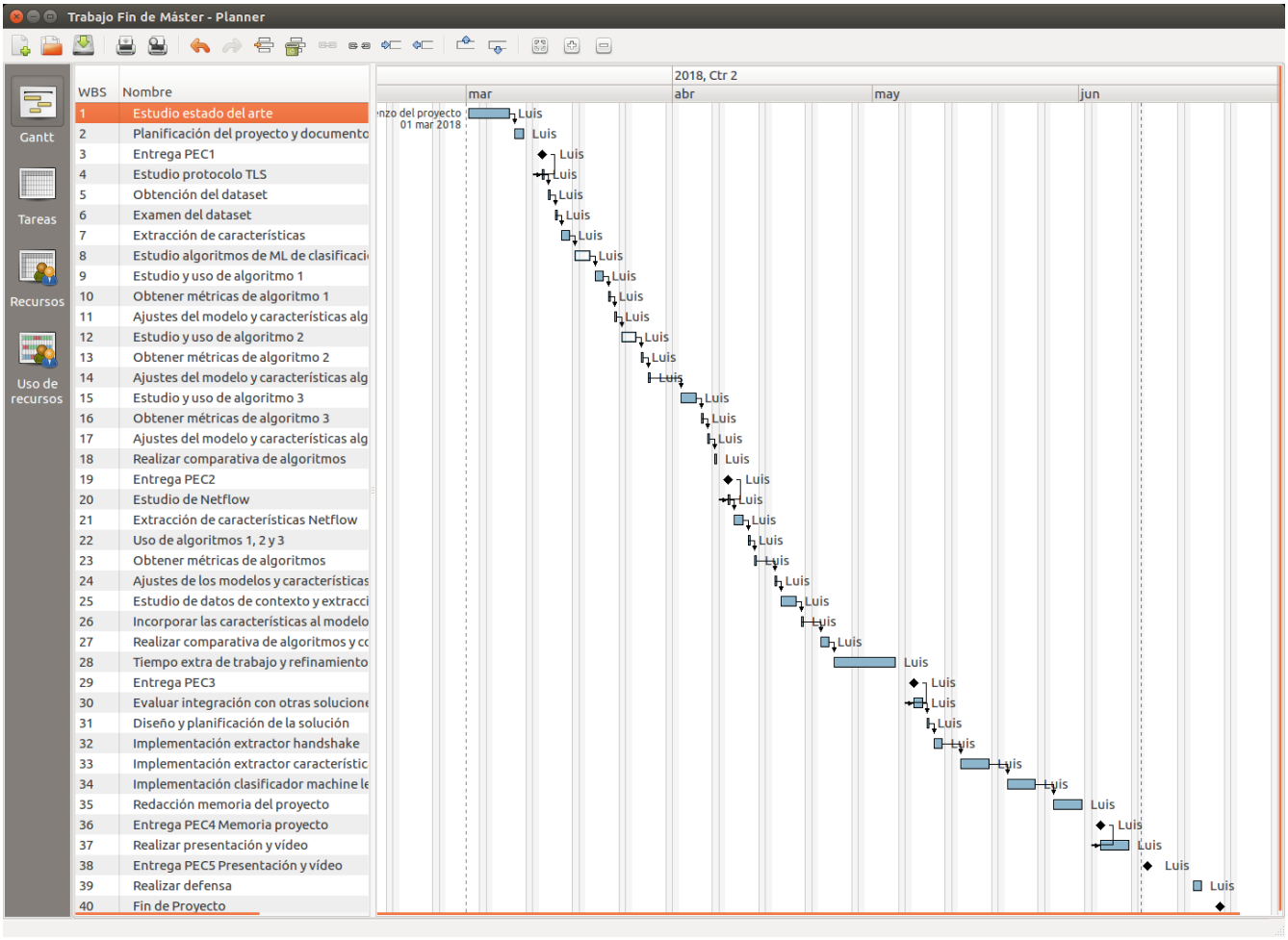

Figura 2. Captura de pantalla del Gantt inicial del proyecto

1.4.2. Hito 1: Clasificador ML con información del handshake (PEC2)

- Estudio protocolo TLS: realizo un estudio del protocolo, RFC, exámen de trazas
- Obtención del dataset: realizar las actividades necesarias para obtener el conjunto de datos sobre los que trabajar
- Examen del dataset: comprobar la información disponible, cómo están etiquetados, etc
- Extracción de características: obtener las herramientas necesarias o desarrollarlas si es preciso para extraer características de los elementos del dataset que correspondan al handshake
- Estudio algoritmos de ML de clasificación: estudiar los algoritmos de clasificación más utilizados para este tipo de problema, las herramientas disponibles y seleccionar 3 algoritmos de estudio
- Estudio y uso de algoritmo 1: estudio del funcionamiento, herramientas y uso del algoritmo con características del dataset
- Obtener métricas de algoritmo 1: realizar pruebas cross-validation y obtener precisión, tiempo de entrenamiento, falsos positivos
- Ajustes del modelo y características algoritmo 1: probar valores de parámetros del algoritmo y la eficacia de quitar/poner características
- Estudio y uso de algoritmo 2: estudio del funcionamiento, herramientas y uso del algoritmo con características del dataset

- Obtener métricas de algoritmo 2: realizar pruebas cross-validation y obtener precisión, tiempo de entrenamiento, falsos positivos
- Ajustes del modelo y características algoritmo 2: probar valores de parámetros del algoritmo y la eficacia de quitar/poner características
- Estudio y uso de algoritmo 3: estudio del funcionamiento, herramientas y uso del algoritmo con características del dataset
- Obtener métricas de algoritmo 3: realizar pruebas cross-validation y obtener precisión, tiempo de entrenamiento, falsos positivos
- Ajustes del modelo y características algoritmo 3: probar valores de parámetros del algoritmo y la eficacia de quitar/poner características
- Realizar comparativa de algoritmos: realizar las tablas comparativas de eficiencia obtenidas por los diferentes algoritmos

1.4.3. Hito 2: Clasificador ML información del flujo (PEC 3)

- Estudio de Netflow: realizo un estudio del protocolo y herramientas
- Extracción de características Netflow: extraer características de flujos netflow
- Uso de algoritmos 1, 2 y 3: uso de los algoritmos con las nuevas características
- Obtener métricas de algoritmos: realizar pruebas cross-validation y obtener precisión, tiempo de entrenamiento, falsos positivos
- Ajustes de los modelos y características: probar valores de parámetros de los algoritmos y la eficacia de quitar/poner características
- Estudio de datos de contexto y extracción de características: obtener datos de contexto (conexiones previas desde misma ip, etc) y extraer características
- Incorporar las características al modelo y obtener métricas: realizar pruebas cross-validation y obtener precisión, tiempo de entrenamiento, falsos positivos
- Realizar comparativa de algoritmos y conjuntos de características: realizar las tablas comparativas de eficiencia obtenidas por los diferentes algoritmos
- Estudio de modelar de comportamiento: revisar trabajo stratosphere IPS y ver si es posible incorporar ideas a nuestro modelo
- Semana extra de trabajo y refinamiento del modelo

1.4.4. Hito 3: Implementación de solución NIDS y entrega de memoria (PEC 4)

- Evaluar integración con otras soluciones y alternativas: evaluar la posibilidad de implementar plugin de snort o usar otras herramientas ya existentes
- Diseño y planificación: diseñar la arquitectura de la solución y planificar
- Implementación de extractor de características del handshake
- Implementación de extractor de características de flujo
- Implementación del clasificador machine learning

- Redacción de la memoria

1.4.5. Hito 4: Entrega de presentación y vídeo (PEC 5)

- Realización de presentación y vídeo

1.4.6. Hito 5: Fin de proyecto

- Realizar la defensa

1.5. Breve sumario de productos obtenidos

Junto con la presente memoria se entregarán los siguientes productos:

- **luIDS**: código fuente del software IDS desarrollado y sus componentes (tlsarchiver, tlscheckmalware y luids).
- **luidsbrain**: código fuente de módulo con los modelos ML para OpenCPU.
- **r-tls**: código fuente R con los módulos de generación de características, extracción, creación de modelos y validación.
- **deploy_luids**: scripts que despliegan el sistema luIDS sobre una máquina virtual como la planteada en el despliegue.
- **mcpfdown**: scripts desarrollados para la descarga de datasets mcfp.
- **dataset_utils**: scripts varios utilizados para la gestión de los datasets.

1.6. Breve descripción de los otros capítulos de la memoria

En el siguiente capítulo Estado del arte, describiré brevemente el estado actual de la tecnología en el ámbito que nos ocupa.

En el capítulo Desarrollo del proyecto, describiré el desarrollo del proyecto.

En el capítulo Resultados mostraré los resultados obtenidos por los modelos de ML.

En el capítulo Conclusiones y futuro enumearé las conclusiones obtenidas del trabajo y las futuras acciones.

Chapter 2. Estado del arte

En este capítulo realizaré un breve repaso al estado del arte que abarca este proyecto. Básicamente consiste en unos resúmenes breves, sin entrar en detalle, ya que para eso están las referencias que allí se indican. A lo largo de las secciones se tratan los tres aspectos fundamentales: sistemas IDS, Machine Learning y el protocolo TLS. Finalmente recopilo una serie de trabajos previos de otros autores que son de interés para su estudio.

2.1. Sistemas de detección de intrusos

La solución que trataré de realizar en este proyecto se encuentra recogida dentro de lo que ampliamente se denominan sistemas de detección de intrusos o IDS. Tal y como se detalla en [7], existen diversos tipos, pero generalmente se clasifican en tres tipos:

- **Host-based IDS o HIDS**: este tipo de sistemas se despliegan en los hosts y monitorizan sus características y los eventos ocurridos en el host, como identificadores de procesos y las llamadas al sistema que hacen para buscar actividad sospechosa.
- **Network-based IDS o NIDS**: este tipo de sistemas se basan en la monitorización del tráfico de red de un segmento de red particular y analiza los protocolos de red, transporte y aplicación para identificar actividad sospechosa.
- **Distribuido o Hybrid IDS**: combina la información de un número de sensores, a menudo de ambos tipos, en un analizador central que es capaz de identificar mejor y responder a una actividad de intrusión.

En general, los IDS tienen dos tipos de aproximaciones para la detección de intrusos:

- **Detección mediante firmas o heurística**: en este tipo se utilizan patrones conocidos de datos (firmas) o reglas de ataque (heurística) que son comparadas con el actual comportamiento para decidir si se trata de una intrusión. Este tipo de aproximación sólo puede identificar ataques conocidos para los que ya existen patrones o reglas.
- **Detección de anomalías**: requiere la colección de datos relativos al comportamiento de usuarios legítimos durante un periodo de tiempo. El comportamiento observado es analizado para determinar con un alto nivel de confianza si el comportamiento proviene de un usuario legítimo o se trata de un intruso.

En lo que respecta a los sistemas de detección de anomalías hay una clasificación comúnmente usada:

- **Estadística**: se basan en el análisis de el comportamiento observado usando métricas de modelos estadísticos univariante, multivariante o series de tiempo.
- **Basados en el conocimiento**: se basan en el uso de sistemas expertos que clasifican el comportamiento observador de acuerdo a un conjunto de reglas de un comportamiento legítimo.
- **Machine-learning (ML)**: se basan en generar automáticamente un modelo de clasificación de acuerdo a datos de entrenamiento obtenidos usando técnicas de *data minning*.

Por lo tanto, a partir de la clasificación de definida, nuestro producto se tratará de un sistema NIDS basado principalmente en la detección de anomalías usando un sistema de detección ML.

2.2. Machine Learning

El Machine Learning o en su traducción "aprendizaje automático" es una rama de la Inteligencia Artificial que permite a los ordenadores aprender de manera autónoma a partir de datos.

Para ello se hace uso de *algoritmos* que dado un conjunto de datos son capaces de determinar relaciones matemáticas entre los atributos y las instancias. Entenderemos por *atributo* a una característica de cada *instancia* y a esta última como una muestra que contiene varios atributos. El resultado de aplicar dichos algoritmos son los *modelos*, que será capaz de determinar los aspectos más relevantes encontrados en los datos con el objetivo de establecer relaciones y patrones. Una particularidad importante de dichos modelos es que tendrán *capacidad predictiva,* esto es: podrán aplicarse a nuevos datos.

Existen tres tipos de tareas clave para las cuales se emplean los modelos de ML:

- **Clasificación**: si a las instancias les asociamos una determinada *etiqueta* o *clase* los modelos podrán determinar a qué clase pertenecen nuevas instancias.
- **Regresión**: Si a las instancias les asociamos un determinado valor continuo, los modelos de regresión podrán predecir un valor para nuevas instancias.
- **Agrupamiento** Podría considerarse como una tarea de clasificación, pero se trata de una tarea en la que no existen un conjunto de clases predefinidas, sino que es el propio modelo el que descubre de forma autónoma cómo las instancias se agrupan.

Los *algoritmos* se dividen en dos grandes grupos:

- **Métodos supervisados**: requieren de un conjunto de datos previamente etiquetado.
- **Métodos no supervisados**: los datos no tienen ninguna etiqueta o clasificación previa.

Por lo tanto, **para resolver nuestro problema necesitamos un algoritmo de clasificación supervisado**. Existen muchos algoritmos que cumplen con este requerimiento, pero vamos a seleccionar tres de los algoritmos más representativos:

Árboles de decisión o Decission tree

Los modelos creados por este algoritmo son modelos con poca capacidad predictiva si los comparamos con los generados por otros algoritmos. Sin embargo es muy utilizado debido a que ofrece una gran capacidad explicativa. Existen dos tipos de árboles de decisión: árboles de clasificación y árboles de regresión. Los primeros se encargan de predecir una variable categórica (una clase) mientras que los segundos predicen una variable continua.

El algoritmo trata de subdividir el espacio de datos de entrada para generara regiones en las que todos los elementos que pertenezcan a una región sean de la misma clase. Si una región contiene datos de diferentes clases es nuevamente subdivida en regiones más pequeñas hasta que se haya particionado todo el espacio de entra en regiones disjuntas que solamente contienen elementos de una misma clase.

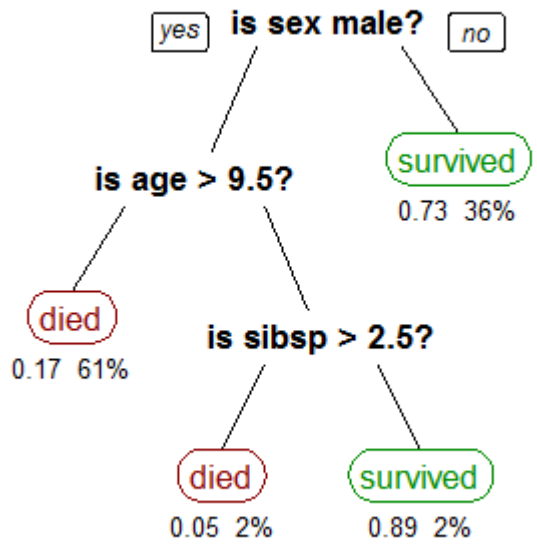

Figura 3. Ejemplo de árbol de decisión

Máquinas de Soporte Vectorial o SVM

Son capaces de resolver problemas de clasificación tanto lineales como no lineales. Es un algoritmo eficiente y con buenos resultados en comparación con otros algoritmos en campos como el reconocimiento de textos y habla, series temporales, estudios de márketing, diagnósticos, etc.

El *SVM* en el problema de clasificación binaria, toma el conjunto de datos de entrenamiento y construye un hiperplano (separador lineal) capaz de dividir los puntos en dos grupos. Si la frontera definida entre dos regiones con elementos de clases diferentes es compleja, en lugar de construir un clasificador complejo que produzca dicha frontera, lo que se intenta es "doblar" el espacio de datos en un espacio de mayor dimensionalidad de forma que con un único corte se puedan separar fácilmente ambas regiones.

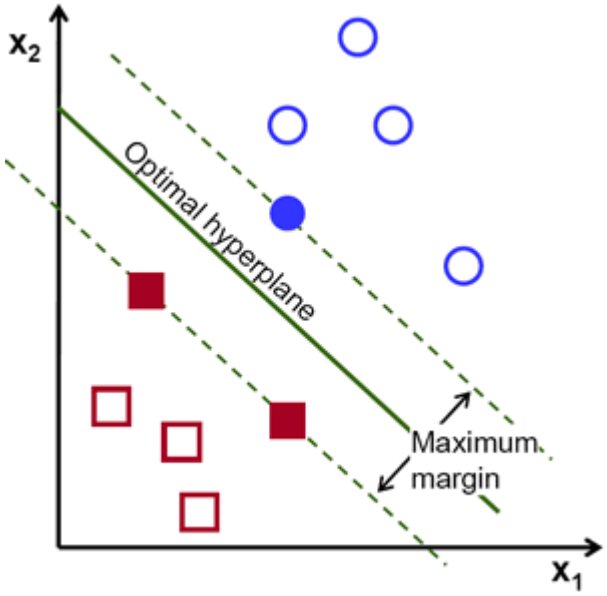

Figura 4. Ejemplo de SVM

Naïve Bayes

Este algoritmo se basa en el concepto de probabilidad condicional y busca maximizar la verosimilitud del modelo, otorgando mayor importancia a aquellos eventos que son realmente relevantes en el juego de datos.

En general, los métodos estadísticos suelen estimar un conjunto de parámetros probabilísticos, que expresan la probabilidad condicionada de cada clase dadas las propiedades de un ejemplo. A partir de aquí, estos parámetros pueden ser combinados para asignar las clases que maximizan sus probabilidades a nuevos ejemplos.

Métricas de clasificación

Existen algunas métricas que se suelen emplear para evaluar el rendimiento de un modelo de clasificación:

Acrónimo	**Descripción**
ERR	Error de clasificación (*missclassification error*)
ACC	Exactitud (*accuracy*)
TPR	Tasa de verdaderos positivos (*true positive rate*)
FPR	Tasa de verdaderos negativos (*false positive rate*)
PRE	Precisión (*precision*)
REC	Recall (*recall*)
SEN	Sensibilidad (*sensitivity*)
SPE	Especificidad (*specificity*)
F1	Puntuación F1(*F1 score*)

Utilizaremos estas métricas para evaluar nuestros modelos, existe una descripción de las mismas en [6]

2.3. El protocolo TLS

El protocolo TLS es un acrónimo de *Transport Layer Security* y es un protocolo que hereda del protocolo SSL que a su vez es un acrónimo de *Secure Sockets Layer*. Ambos son protocolos criptográficos que proporcionan servicios de comunicación segura.

La última versión del protocolo TLS es la `1.2` [8], aunque ya se encuentra en borrador la versión `1.3`. El protocolo TLS `1.2` proporciona compatibilidad hacia atrás con las versiones de TLS y SSL (ej TLS `1.0`, `1.1` y SSL `3.0`). El protocolo TLS `1.2` opera sobre un protocolo orientado a conexión (normalmente TCP), pero existe una variante para protocolos orientados a datagrama (UDP) basada en TLS llamada *Datagram Transport Layer Security* (DTLS) definida en [9]. En este estudio nos centraremos exclusivamente en TLS `1.2`.

Los servicios de seguridad que TLS ofrece están basados en diversas suites criptográficas que usan certificados provistos por una *Infraestructura de Clave Pública* (PKI). A pesar de esto TLS puede también establecer conexiones seguras entre pares anónimas (obviamente sin garantizar la veracidad de las partes). Como dije, TLS está sobre un protocolo orientado a conexión (como TCP) y se compone de dos capas: la capa de registro y la capa de protocolos.

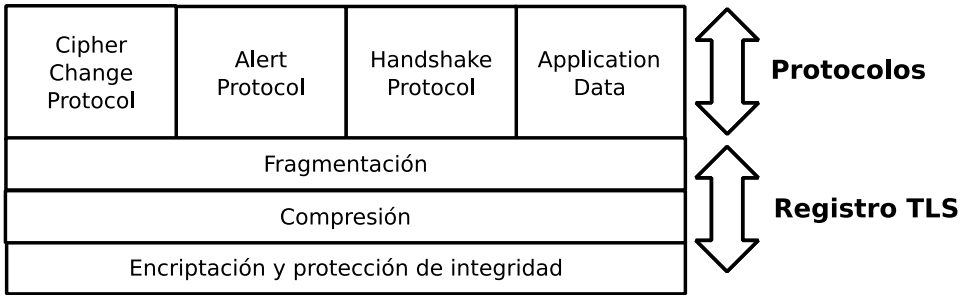

Figura 5. El stack del protocolo TLS

La capa de registro o *Record layer* proporciona servicios de fragmentación, compresión, cifrado e integridad. Sobre dicha capa se sitúan los cuatro protocolos de TLS:

- **Handshake Protocol**: es el protocolo que se utiliza para negociar el intercambio de claves, certificados, protocolos criptográficos, etc. En una sesión, la primera vez que se utiliza este protocolo estará en claro, lo que hace que este protocolo sea fundamental para nuestro estudio. Se compone de diversos mensajes. Siendo los más importantes:
  - `ClientHello`: es el mensaje mediante el cual el cliente se presenta y propone las suites criptográficas que puede utilizar.
  - `ServerHello`: es el mensaje con el que el servidor contesta, presentándose a su vez y seleccionando una suite criptográfica para la conversación.
  - `Certificate`: es el mensaje en el que puede viajar una cadena de certificados y se utilizará para transportarla.
- **Alert Protocol**: es básicamente un protocolo de control. Se utiliza para notificar problemas en la conexión, solicitudes de renegociación de cifrado, cierre de conexión, etc.
- **Cipher Change Protocol**: es el protocolo (realmente se compone de un único mensaje) que se emplea para notificar que a partir de ese mensaje se cambian los mecanismos de cifrado por los negociados previamente.
- **Application Data**: es el protocolo sobre el que se encapsulan los mensajes del nivel de aplicación sobre el cual el protocolo TLS dará servicio.

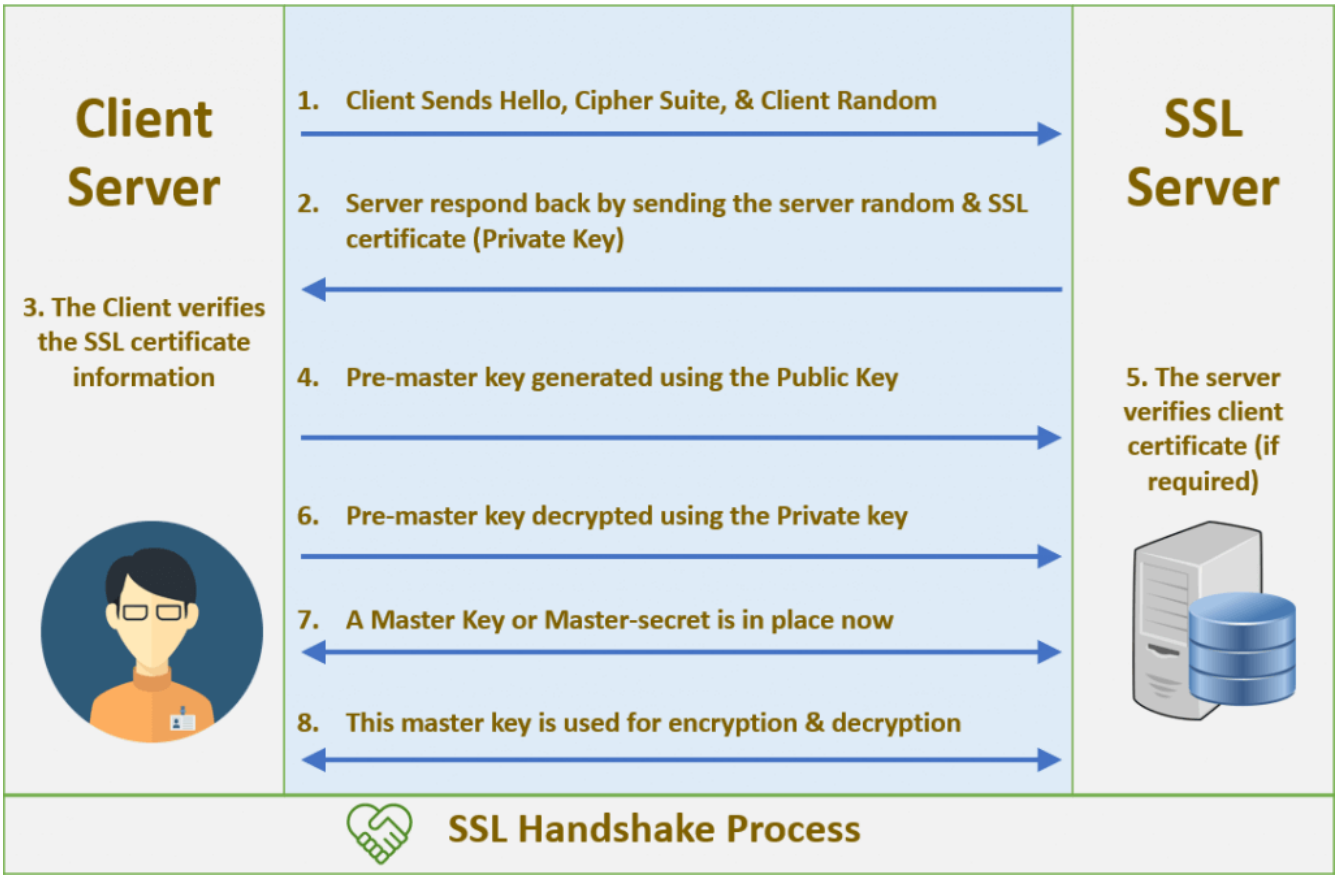

Figura 6. Proceso de realización de un handshake, fuente [cheapsecurity]

El protocolo admite la restauración de sesiones, optimizando de este modo su uso práctico y admite extensiones al protocolo. Por nombrar las más utilizadas:

- **Application Layer Protocol Negotiation (ALPN)**: permite indicar el protocolo de nivel aplicación que se utilizará en la conversación.
- **Eliptic Curve Capabilities**: definido en [11] permite comunicar los tipos de parámetros que acepta para *algoritmos de curvas elípticas*.
- **Secure Renegotiation**: mejora la seguridad en la función de renegociación de TLS.
- **Server Name Indication (SNI)**: provee un mecanismo al cliente para especificar el nombre del servidor al que se desea conectar (esto permite la existencia de diferentes nombres de servidor sobre una misma IP).
- **Session Tickets**: introduce un nuevo mecanismo de restauración de la sesión que no requiere almacenamiento del lado del servidor, sino que funciona como una especie de cookie.
- **Signature Algorithms**: habilita al cliente comunicar qué algoritmos de hashing soporta para utilizar alguno de estos en lugar del algoritmo de firma por defecto `SHA1`.
- **OCSP Stapling**: permite proporcionar al cliente una prueba reciente de información de revocación del cifrado para evitar que éste tenga que comprobarla.

2.4. Trabajos previos

El uso de machine-learning se ha utilizado tanto en sistemas HIDS como NIDS [12]. En sistemas NIDS ya se ha utilizado anteriormente para clasificar tráfico e identificar malware [13] sobre conexiones no cifradas, implementando sistemas supervisados e incluso semi-supervisados

integrados con IDS en [14], llegando a obtener una precisión del 98,88% y falsos positivos de 0,55%. Existen otros papers en los que comparan el rendimiento de IDS, su precisión e integración como en [15].

Sin embargo, en la actualidad el tráfico cifrado está creciendo y obviamente también lo están incorporando todo el nuevo malware, ya que esto dificulta mucho su detección. El protocolo más empleado en este ámbito es el protocolo TLS, que cifra a nivel capa de transporte y permite y es el empleado por numerosos protocolos de nivel aplicación como HTTP, SMTP, IMAP, etc. Son muy interesantes en este ámbito los papers de Blake et. al [16] [17] [18], en los que se introducen numerosas técnicas para el análisis de este tipo de conexiones, entre las que se encuentra la extracción de datos no cifrados (el handshake), metadatos de flujos, tamaños, conversaciones con el dns, etc y obteniendo unas tasas de precisión del 99,99% y una tasa de falsos positivos de casi el 0%. Otra fuente muy interesante en este ámbito es todo lo publicado por Stratosphere IPS [20], que modelan el comportamiento de las conexiones y ayudan a detectar tráfico de tipo C&C de las botnets [19].

Chapter 3. Desarrollo del proyecto

En este capítulo es el más extenso ya que describo el desarrollo del proyecto. Como comento en la Introducción, el proyecto sigue una metodología iterativa, por lo que se han ido introduciendo elementos en las diferentes iteraciones. Sin embargo, en pro de la legibilidad describo las fases del proyecto de una forma secuencial y ordenada.

3.1. Análisis del problema

Tras la definición de objetivos y planificación vistos en la Introducción, es fundamental dibujar un escenario en el que contextualizar al producto final y poner en orden todos los conceptos e ideas extraídos a lo largo de mi carrera profesional en áreas de redes y sistemas junto con toda la documentación y trabajos estudiados durante del Estado del arte. Y, a partir de esto, formular una serie de ideas e hipótesis que puedan ayudar en la implementación del clasificador.

3.1.1. El producto final

En esencia, el producto final que debemos construir se trata de una herramienta que se situará en algún sistema intermedio con funciones de enrutamiento o en algún *puerto espejo* del mismo, que analizará de forma pasiva las conexiones cifradas y que, usando modelos de *ML*, clasificará dicha conexión en función de un determinado valor de probabilidad.

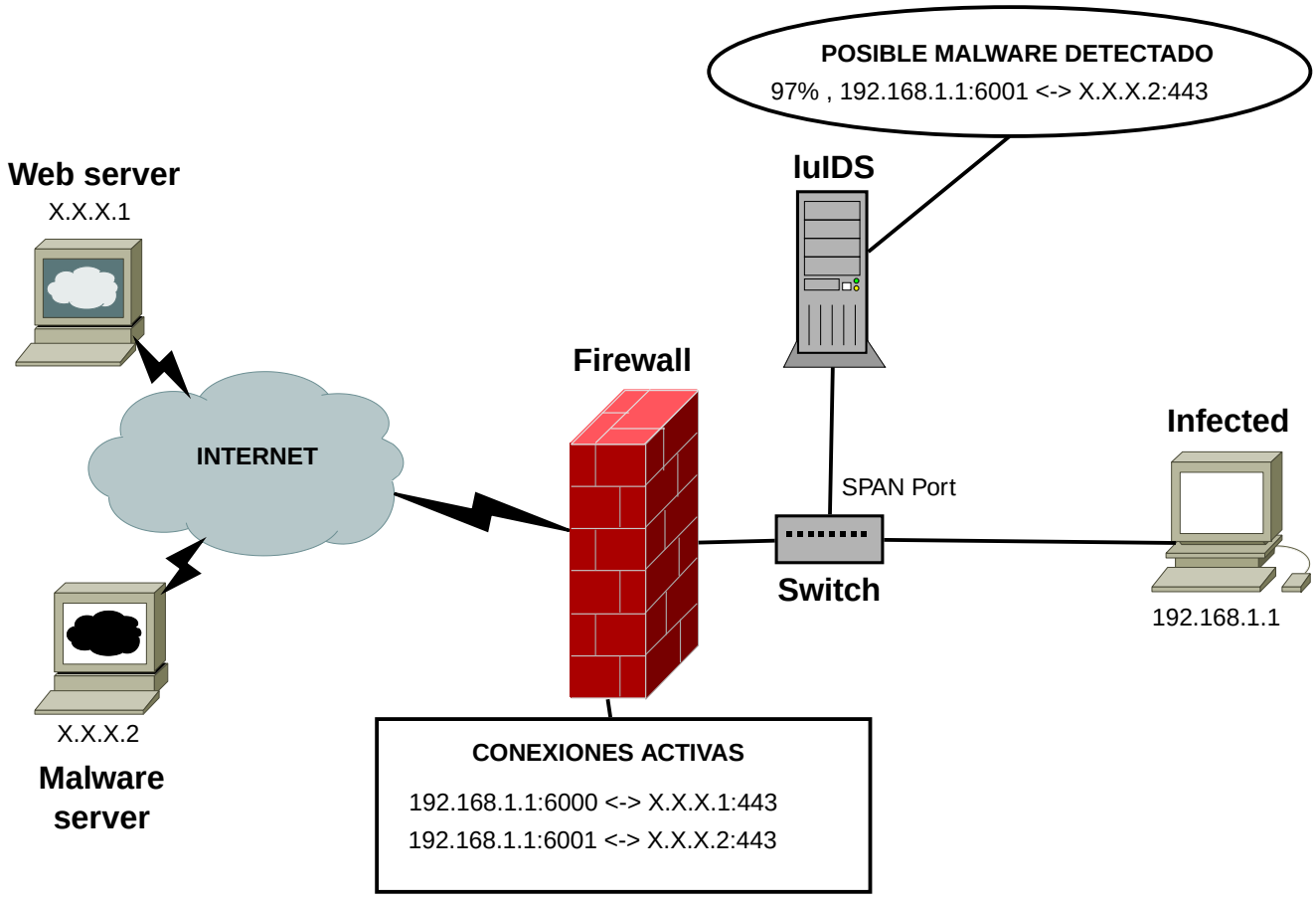

Figura 7. La herramienta luIDS

Tras evaluar varios nombres decidí llamar finalmente luIDS al producto final.

3.1.2. Hipótesis e ideas para la clasificación

Una conexión TCP no es más que un par de flujos de datos, cada uno en una dirección, establecidos por dos procesos residentes en dos sistemas diferentes. El uso del protocolo TCP garantizará que todos los mensajes se entregan (o al menos se conocerá si no se entregan) y que lo harán en el orden en que se enviaron. Para realizar esta conexión TCP los procesos simplemente harán las llamadas al sistema necesarias y será el sistema operativo el que se encargue de todo.

Sin embargo, la labor del sistema operativo se queda ahí: en la gestión de las conexiones TCP. El protocolo TLS deberá ser implementado por la propia aplicación (o por una biblioteca especializada) y proporcionará servicios de autenticación, integridad y cifrado al protocolo de aplicación que se emplee "por encima" (*HTTP*, *LDAP*, etc). Por lo tanto, serán los procesos los encargados de, sobre los servicios TCP ofrecidos por el sistema operativo, establecer la sesión TLS y a continuación implementar sobre dicha sesión el protocolo de aplicación para el que se desean los servicios de autenticación, integridad y cifrado.

En base a esto podemos definir dos aspectos clave que son responsabilidad de los programas que dan origen a los procesos:

1. Implementación de TLS: establecimiento y gestión la sesión TLS.
2. Implementación del protocolo de aplicación usado.

Llegados a este punto, nos planteamos la cuestión fundamental: ¿cómo podemos clasificar con cierto grado de certeza que el proceso que está detrás de una conexión es una instancia de un malware y no de un programa legítimo? Para tratar de responder a esta cuestión nos haremos una serie de preguntas acerca del comportamiento del malware y las responderemos con una serie de hipótesis con el fin de ayudarnos en la difícil tarea de la clasificación.

¿Como se presenta?

Como dije, corresponde a la aplicación la implementación del protocolo *TLS*. A la hora de implementar dicho protocolo, el malware tiene dos opciones:

- Implementarlo él mismo o utilizar una implementación muy reducida o específica para este caso de uso.
- Utilizar bibliotecas estándar o del sistema.

La primera de las opciones tiene una serie de ventajas para los creadores de malware, ya que pueden incluir de manera estática con el ejecutable manteniendo un tamaño reducido, controlan por completo el proceso de conexión, evitan posibles incompatibilidades con versiones instaladas en el sistema objetivo, evitan la distribución de ficheros `.dll` y `.so`, evitan controles que pudieran existir en dichas bibliotecas, evitan configuraciones de proxies, etc.

Por todas las razones enumeradas existe un incentivo claro por parte de los desarrolladores de malware a que opten por esta opción. Sin embargo, abren diversas oportunidades para la detección:

Implementación incorrecta

- El malware emplea un puerto reservado pero no usa el protocolo TLS.
- El malware hace como que habla TLS: podría darse el caso de malware que imita el establecimiento de una conexión TLS para saltarse posibles controles. Podría por ejemplo imitar el handshake de una recuperación de sesión e intercambiar un mensaje tipo `ChangeCipherSpec` y emitir paquetes con la cabecera de *TLS* de tipo aplicación pero con un payload que nada tenga que ver.
- El malware no implementa correctamente el handshake: podría ser que la secuencia de handshake no sea válida, tenga algún error, defina una extensión no existente, etc.

Es posible realizar heurísticas como la existencia de la cabecera, comprobar que la secuencia es correcta, existe una sesión previa, etc. Además se podría comprobar también que la distribución de bytes del payload supuestamente cifrado no sigue una distribución homogénea, lo que indicaría que no está cifrado.

Implementación incompleta

- El malware usa criptografía débil u obsoleta: es más fácil de implementar, tiene menos coste computacional, etc.
- El malware no mantiene la persistencia de sesiones: un elemento del protocolo muy interesante e imprescindible para su éxito es la posibilidad de restaurar sesiones para no tener que renegociar el intercambio de claves de sesión. Puede existir malware que, por simplicidad, no implemente este mecanismo.
- El malware no utiliza fragmentación o mensajes múltiples del protocolo de handshake: es posible que nuevamente, por simplicidad, el malware no implemente estas características.
- El malware no implementa extensiones: las implementaciones del malware no hacen del uso de extensiones ampliamente utilizadas y negociadas por defecto por gran parte de aplicaciones lícitas. Por ejemplo SNI, ALPN, etc.

La otra opción que podemos encontrar es que haga uso de librerías del sistema o estándar compartidas por muchas aplicaciones legítimas. En este caso, todas las oportunidades enumeradas anteriormente desaparecen, limitándose a que:

- El malware utilice protocolos de cifrado y firma débiles o raramente utilizados por aplicaciones que hagan uso de la misma biblioteca.
- El malware defina una configuración particular, haciendo uso de extensiones o valores raramente empleados.

A partir del "cómo se presenta" es posible tratar de hacer un *fingerprint* de aplicaciones y de librerías de aplicación que implementan TLS. Este fingerprint además de servir para crear nuestro modelo, podrá utilizarse como complemento a otros mecanismos de clasificación.

¿Con quién habla?

El siguiente elemento a tener en cuenta es quién está al otro lado de la conexión. La primera idea que surge en este aspecto lógicamente es… quien está al otro lado, ¿es de fiar?, o como dice el

refranero popular... "dime con quien andas y te diré quien eres".

Podría hacerse uso de servicios de *blacklist* con bases de datos de *IPs* comprometidas e incluso utilizar esta información en caso de positivo para etiquetar y entrenar a nuestro modelo. Esto también ofrece otras oportunidades como la implementación de mecanismos de *blacklist* internos generados a partir de hallazgos, de modo que si otra IP de la red que queremos monitorizar inicia conexión con una IP para la que hemos detectado previamente una incidencia de malware, podamos alertar al usuario de que dicho sistema podría estar también comprometido.

Aunque la información de *blacklist* podría sernos útil en una implementación real, esta información difiere a lo largo del tiempo y, dado que los dataset de los que disponemos son de hace años, no podemos utilizarla para nuestro estudio.

Además de la IP, tenemos que tener en cuenta que detrás de esa IP en el puerto definido existe un proceso a la escucha. Dicho proceso puede ser:

- Malware con su propia implementación o de bibliotecas estándar
- Un servidor de aplicaciones estándar con recursos (apliación malware, ficheros, etc)

En ambos casos podemos aplicar lo visto en el apartado anterior, con la particularidad de que el mensaje de saludo del servidor depende de las opciones ofrecidas por el cliente.

¿Cómo lo ha encontrado?

En este aspecto, podemos definir la hipótesis de que la inmensa mayoría de las aplicaciones legítimas hace uso de nombres y no de IPs y, por lo tanto, hacen uso del protocolo *DNS* para resolverlas. Esto abre una serie de oportunidades:

- Si se establece una conexión TLS con una IP no resuelta previamente podemos considerarlo sospechoso.
- Si el nombre resuelto asociado a la IP está en una *blacklist* podemos proceder del mismo modo que en el apartado anterior.
- El nombre resuelto... ¿coincide con el del certificado negociado en el handshake y el SNI de la petición?. En caso negativo, podríamos considerar este hecho como muy sospechoso.
- Cuestiones sobre el nombre resuelto: ¿cómo es el dominio? ¿tiene mala reputación? ¿el nombre es muy raro?. En todos los casos se podría utilizar esta información para establecer una métrica para añadir al modelo.

Para tomar ventaja de las oportunidades enumeradas, nuestra herramienta también debería procesar y almacenar las consultas y respuestas del protocolo DNS.

Quien habla... ¿es quien dice ser?

Ya he avanzado en el apartado anterior que la información DNS puede suponer una gran ventaja,

especialmente en la comprobación de la petición SNI y que el certificado valide para el nombre para el que se hizo la petición.

En lo que respecta a los certificados podemos asumir que algunos malware:

- Utilizan certificados autofirmados o incluso "generados al vuelo" (las fechas de validez y de caducidad nos pueden ayudar).
- Utilizan mecanismos de cifrado y firma débiles (los algoritmos y las longitudes de clave nos pueden ayudar).
- Tienen más propensión a utilizar entidades de certificación gratuitas como "Let's Encrypt" [4].

Además de las comprobaciones rutinarias de certificados (nombre válido, fecha, certificado no revocado), se podría crear un mecanismo o servicio de ponderación de Entidades de Certificación que estimase un valor o clasificación en función de unos criterios de confianza (tal vez esto ya exista, no lo he comprobado). De este modo se podría tener en cuenta este valor a la hora construir nuestro modelo.

Como ocurre con las blacklist, la validez de certificados y de su revocación dependerá del tiempo. Esto hace que no se pueda emplear la información de revocación para un estudio de capturas antiguas. Pero existe un problema adicional que trataré más adelante, y es que TLS implementa la restauración de sesión, lo que evita el intercambio de certificados y la generación de claves de sesión. Esto implica tener que almacenar identificadores de sesión de las conexiones para poder relacionarlos con los certificados que se usaron.

¿Cómo se comporta?

Finalmente el protocolo TLS no es más que una capa que garantiza una serie de servicios a los protocolos de aplicación. Dichos protocolos tendrán una forma de comportarse diferente en dos grandes líneas.

Flujo de datos

Aquí partimos de la hipótesis de que el flujo de datos será diferente entre aplicaciones, por ejemplo podemos asumir que una descarga de una página web usando HTTP será diferente de un malware tipo *Botnet* hablando con alguno de sus servidores *C&C*. Pero tengamos en cuenta que esto ocurre también entre aplicaciones legítimas: no será igual un flujo HTTP que uno *HTTP/2* que multiplexa las peticiones. Por si esto fuera poco, podrán existir diferencias incluso aunque use dicho protocolo entre una aplicación tipo *Dropbox* y un navegador. Esto da lugar a la siguiente idea:

Si el flujo de datos observado no encaja con el modelo del protocolo y servicio "esperado", podremos estar ante un conexión proveniente de un proceso de malware.

Aquí surge una nueva cuestión, ¿cuál es el servicio "esperado"?, ¿cómo podemos saber cuál es el modelo más adecuado para contrastarlo?. Para ello podemos utilizar las siguiente ideas:

- El puerto destino generalmente nos ofrece información sobre los protocolos de nivel aplicación

que harán uso de los servicios de TLS.

- Si en la petición se hace uso de extensiones que negocian el protocolo en claro (como ALPN), podremos usar dicha información para seleccionar el modelo.
- El fingerprinting que podamos realizar del mensaje `ClientHello` del handshake.
- Por la información DNS y de los certificados. Por ejemplo, si conocemos que las conexiones a Dropbox emplean unos nombres de servidor y certificados determinados, podremos utilizar dicho modelo para contrastarlo.

Número de conexiones y recurrencia

Del mismo modo que hicimos en el punto anterior, podemos partir de la hipótesis de que el número de conexiones que se realizarán contra un determinado servidor o servidores así como la recurrencia de las mismas variará entre aplicaciones.

Por poner el mismo ejemplo, el protocolo HTTP tendrá más conexiones que HTTP/2 ya que este último multiplexa pero ambos cerrarán las conexiones una vez alcanzado el tiempo definido por el navegador. Sin embargo, es posible que un Botnet mantenga abierta una sesión de estado durante mucho más tiempo o incluso empiece a hablar con cientos de direcciones a la vez. En lo que respecta a la recurrencia, el patrón de navegación de una persona será a ráfagas mientras que el malware o los actualizadores de aplicaciones y sistema operativo lo hará de forma recurrente y periódica.

Si el número de conexiones observado y la recurrencia de las mismas no encaja con el modelo del protocolo y servicio "esperado", podremos estar ante un conexión proveniente de un proceso de malware.

En este punto podremos utilizar las mismas técnicas descritas en el punto anterior.

Aunque en este trabajo centro los esfuerzos en explorar el uso técnicas de ML para la detección, como he tratado de mostrar, podrían tener más efectividad si van de la mano con determinadas heurísticas. Esto obviamente está pendiente de demostrar y se deja para trabajos futuros.

3.1.3. Proceso de construcción y uso de modelos

Para finalizar con el análisis, definí el proceso que seguiría para la construcción de los modelos y cómo el sistema IDS haría uso de ellos.

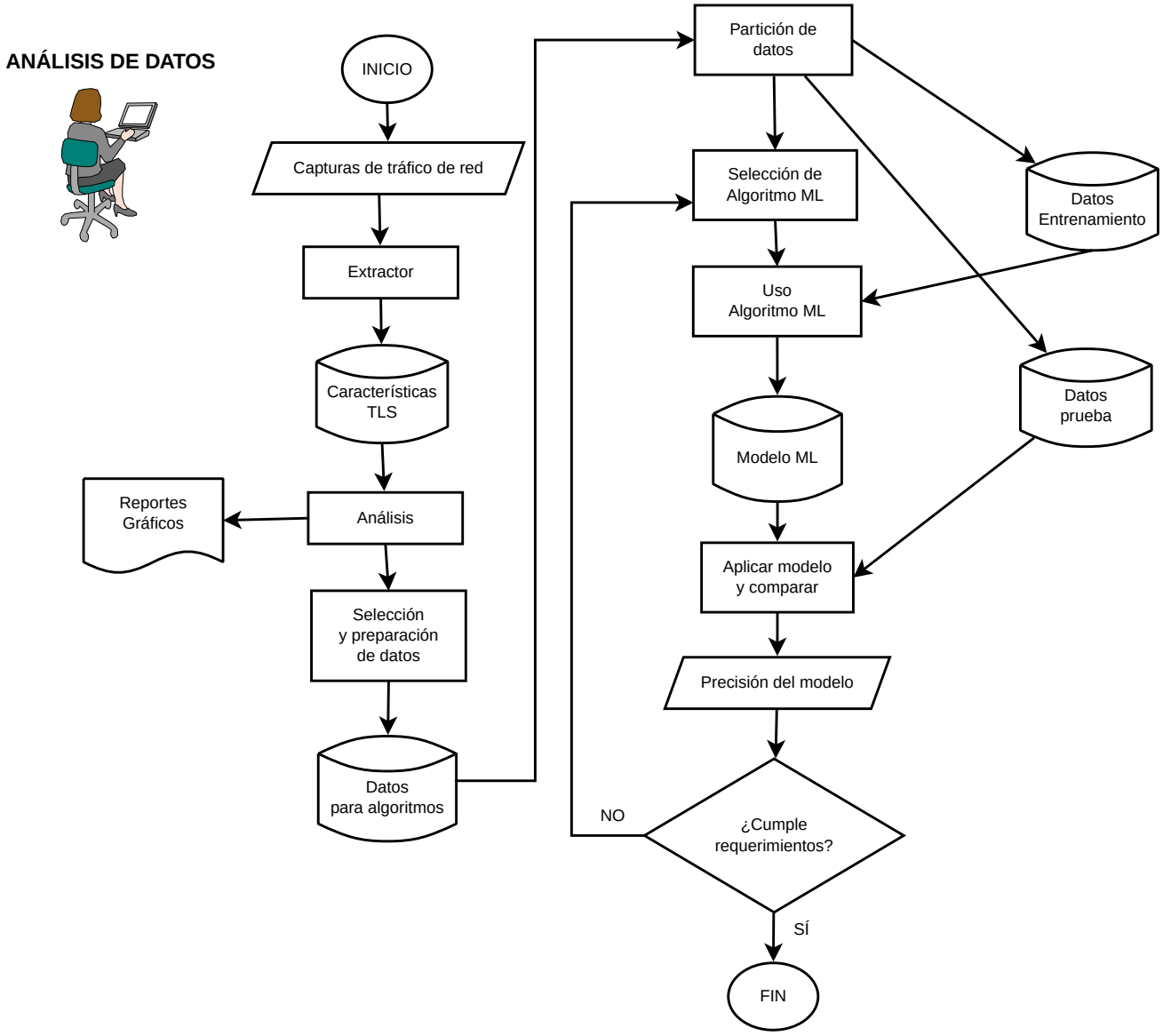

Figura 8. Flujo del construcción de los modelos

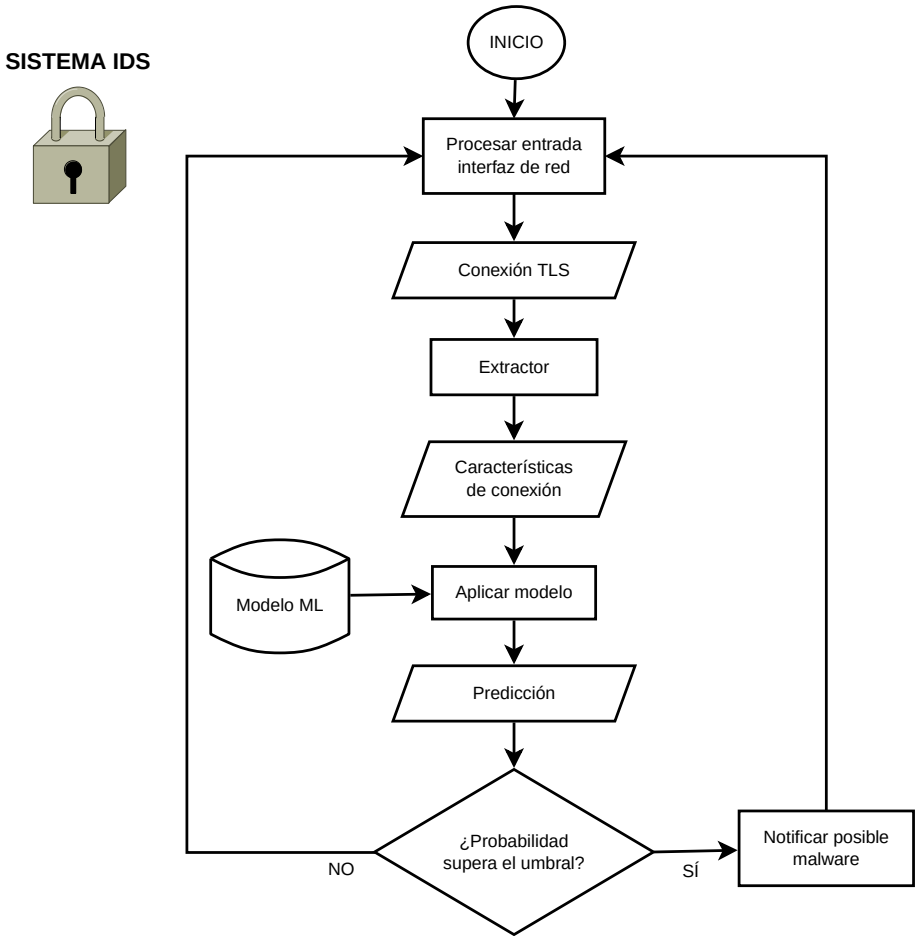

Figura 9. Flujo de datos del uso de los modelos

3.2. Obtención de los datos

Una vez completado el análisis previo del problema, el siguiente paso es la obtención del conjunto de datos sobre el que trabajar.

3.2.1. Selección y descarga

Como ya dije al comienzo, aunque en un principio iba a disponer de un dataset o algún tipo de herramienta para generarlo por parte de la empresa a la que iba a estar vinculado este proyecto, finalmente no tuve acceso a ninguno. Por ello finalmente opté por usar dos de los que descubrí durante el estado del arte asociados al proyecto Stratosphere IPS [20]:

- **CTU-13** [21]: es un dataset que contiene tráfico de 13 diferentes tipos de botnet. El dataset incluye capturas que contienen únicamente el tráfico de botnets, información de netflows etiquetados y el ejecutable del malware.
- **Malware Capture Facility Project** [22]: en este proyecto se han creado una gran cantidad de datasets disponibles públicamente y que contienen tráfico normal, malware y mezclado. Estos datasets contienen mucha más información que la proporcionada en CTU-13, ya que no sólo se limitan a las capturas y los netflows, también incluyen logs de diversas herramientas, como *broIDS* y *argus* y en muchos casos utilizan un proxy mitm para tratar de desencriptar el tráfico del malware. El conjunto de herramientas utilizadas se describen en la web del proyecto.

El término dataset es un término genérico y se refiere a un conjunto de datos que tienen algo en común y se utilizan para su estudio. En el caso de MCFP, denominamos dataset a las capturas y datos asociados a un determinado escenario que suele estar asociado a un tipo específico de malware. Para este proyecto adopto dicha terminología.

Durante la descarga de los datasets del MCFP, aparecieron nuevos problemas: la información era excesiva para descargar, no tenía suficiente disco y con mi ADSL tardaría semanas. Además la información de las capturas era apenas el 25% de las descargas del dataset ya que los datos de logs, argus, etc copaban el resto y a priori no iba a necesitar esta información. Otro aspecto importante fue que en la adquisición de muchos dataset se había utilizado la herramienta *mitmproxy* [23], lo que hacía que no iba a ser necesario descargar todos los datasets.

La herramienta mitmproxy reemplaza el servidor destino para el cliente, por lo que la conversación TLS que queremos estudiar no se realiza contra el servidor del malware sino contra esta herramienta. Este hecho, aparte de distorsionar las observaciones, hace que gran parte del malware no funcione, bien por implementaciones incorrectas o por comprobaciones de dicho malware de los certificados.

Por todos estos problemas con las descargas desarrollé un pequeño software en *shellscript* al que llamé **mcfp downloader**. Este software se encarga de gestionar la descarga de datasets del proyecto MCFP, permitiendo de este modo descargar sólo los ficheros de captura y etiquetar los datasets por alguna etiqueta.

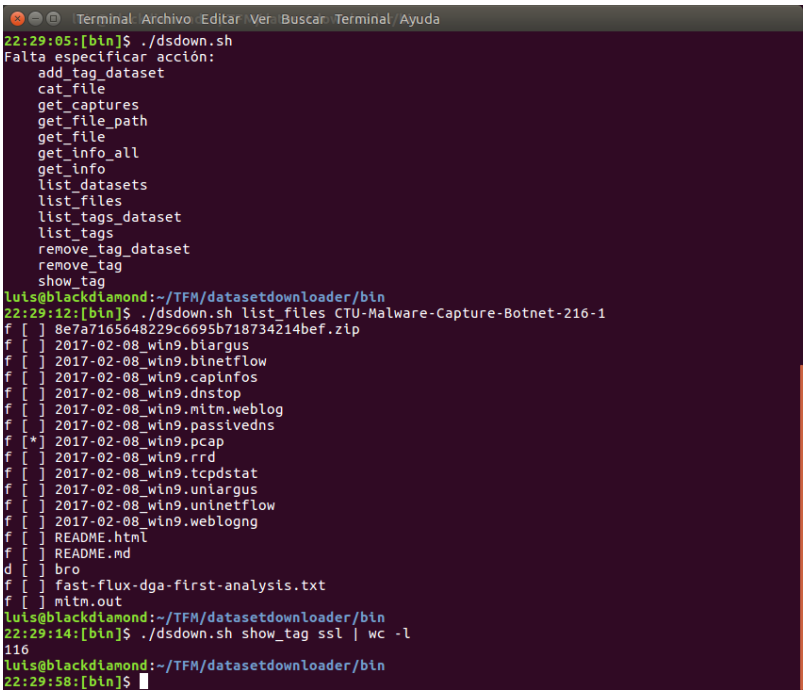

Figura 10. Captura de pantalla de utilidad dsdown.sh

3.2.2. Examen y criba del dataset

Como indiqué en el apartado anterior, uno de los objetivos era no descargar aquellos datasets que utilizasen la herramienta mitmproxy. Para ello, utilizando la herramienta que desarrollé, descargué

los índices y etiqueté todos aquellos que no tuvieran un fichero que contuviese la cadena `mitm`.

El siguiente paso en la criba fue obtener aquel malware que emplease el protocolo TLS en los puertos `443` o `8080`. La elección de este último puerto fue porque encontré tráfico TLS en ese puerto en alguna de las capturas y esto permitiría el estudio de un mayor número de muestras.

Para realizar esta criba, lo que hice fue escribir otros shellscripts que, utilizando las herramientas *tcpdump* [24] y *tshark* [25] me dijeran qué capturas hacían uso del protocolo *SSL/TLS* y así etiquetarlas para usarlas en el análisis posterior. Las que no contenían este tipo de tráfico obviamente se descartaron.

Observar la no existencia de este tipo de tráfico en una gran parte del conjunto de las capturas, nos hace reforzar la hipótesis de que durante las mismas, las máquinas no tenían procesos *no-malware* que hiciesen conexiones TLS. Volveremos sobre esta idea más adelante.

A grandes rasgos el criterio previo de selección de los datasets válidos para el análisis fue el siguiente:

```
## PROCESO DESCARGAR CAPTURAS QUE NO USEN MITM

PARA dataset EN datasets

    SI dataset NO USA mitm ENTONCES
        etiquetar dataset non-mitm
        descargar ficheros capturas
    FIN SI

FIN PARA

## PROCESO DE SELECCIÓN DE DATASETS QUE USEN SSL

PARA dataset EN tiene_etiqueta(non-mitm, datasets)

    comprueba tráfico ssl en captura de dataset
    SI tiene_ssl(captura) ENTONCES
        etiquetar dataset ssl
    FIN SI

FIN PARA
```

Tras el proceso final de criba quedaron los siguientes datasets:

Table 1. Criba en datasets MCFP

Paso	Total	Malware	Mixed	Normal
total	407	378	5	24
non-mitm	227	202	5	20

Paso	Total	Malware	Mixed	Normal
ssl	116	95	5	16

Table 2. Criba en datasets C-13

Paso	Total	Malware	Mixed	Normal
total	13	13	0	0
ssl	5	5	0	0

los scripts que utilizados para la criba se encuentran el producto *dataset_utils*.

3.2.3. Incorporación de datasets propios

Como posteriormente se explicará, en una iteración se generan dos nuevos datasets:

- *N-LUI-001*: captura de equipo Windows 7 actualizándose, y posteriormente navegando con Internet Explorer 9 y 11.
- *N-LUI-002*: captura de un Ubuntu 16.04LTS con Dropbox y Firefox.

3.2.4. Codificación de los datasets

Un dataset lo proporciona un proveedor o un proyecto, que puede ser *MCFP* o yo mismo, tiene un código definido por dicho proveedor y un mecanismo de etiquetado de los flujos. Basándome en dicha información creo la siguiente codificación para los dataset:

```
CÓDIGO DATASET: E-PPP-CCCC
--
E:    Mecanismo de etiquetado
PPP:  Código de longitud fija asignado al proveedor
CCCC: Código de longitud variable definido por el proveedor
```

En nuestro proyecto tenemos 3 mecanismos diferentes de etiquetado: `N` o normal, `I` o infectado y `X` o mezclado. En el caso `N` todas las conexiones del dataset se etiquetan como normales o no-malware, en el caso `I` todas las conexiones del dataset se etiquetan como infectado o malware y en el caso `X` se debe de emplear un mecanismo ad-hoc de clasificación del dataset. Este criterio es extensible a mecanismos de etiquetado futuros.

En el caso de proveedor en nuestro proyecto hay tres códigos: `C13` correspondientes a dicho dataset, `CTU` correspondientes al proveedor `MCPU` y `LUI` son los datasets realizados por mi.

3.2.5. Dataset final

Tras examinar un poco más en detalle los nombres de carpetas y ficheros `README` de los diferentes datasets, se comprueba que todo el malware es de tipo Botnet y no se especifica el tipo de familias al que pertenece, por lo que **se descarta la implementación de un sistema multiclasificador**.

Después de verificar la documentación de los datasets de tipo `Mixed` y verificar las capturas, se

observa que la tarea de etiquetado de las conexiones será bastante compleja ya que habría que basarse en un intervalo de tiempo en el que además concurren conexiones no-malware. Por ello un poco más adelante en el proyecto **se toma la decisión de prescindir de los dataset de tipo `Mixed` para la construcción de los modelos**. Aun así también se procesarán para realizar pruebas.

Más adelante en el proyecto, tras la extracción de características, se verificará que no existen diferencias entre los dataset de C-13 y sus homónimos en MCFP, por lo que **se descartan las capturas del dataset *C-13*** del estudio.

Table 3. Conjunto final de datasets utilizados para la creación de los modelos

Proveedor	**Total**	**Malware**	**Normal**
MCFP	111	95	16
luIDS	2	0	2
Total	113	95	18

3.3. Extracción de características

Una vez obtenido el conjunto de datos, era necesario convertir esa información de las capturas en *formato pcap* a algo que pudiera ser manejable para su análisis.

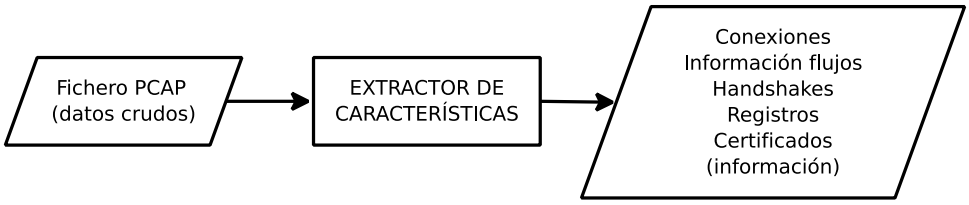

Figura 11. Proceso de extracción de datos de los ficheros PCAP

3.3.1. Dearrollo del extractor

Para realizar la extracción tenía una serie de opciones, que en orden creciente de complejidad/potenicia eran:

1. Usar los logs disponibles en los propios datasets y hacer scripts de parseo
2. Utilizar un mix de herramientas: *tlsfingerprint* [26], *tshark* [25], etc
3. Utilizar Bro IDS [27] y sus posibilidades de scripting
4. Escribir mi propio extractor

Finalmente opté por la cuarta opción. Los motivos fueron los siguientes:

- sería mucho más fácil implementar posteriormente la prueba de concepto del IDS (objetivo principal del proyecto)
- tenía un control total sobre todo el proceso de conversación TLS, lo que me proporcionaría ventajas sobre otros trabajos realizados
- podría fácilmente incorporar chequeos y heurísticas adicionales
- podría fácilmente almacenar y clasificar la información en una base de datos para su posterior

análisis y procesamiento

Antes de comenzar el desarrollo de lleno, realicé un pequeño prototipo que analizaba los handshakes `HelloClient`. Vi que aquello era viable por lo que seguí con la idea de implementar mi propio extractor.

Lo más complejo en desarrollar desde el punto de vista técnico, que era la captura y reordenación y ensamblado del TCP ya estaba implementado por la librería *gopacket* [28]. Sin embargo, existieron varias dificultades que hubo que resolver:

- Tuve que implementar más mensajes y elementos de los que pensaba en un principio.
- Tuve que lidiar con el ensamblado y múltiples mensajes del subprotocolo de handshake
- Me enfrenté al problema de la asociación de flujos para formar las conexiones.

Por ello, el desarrollo del extractor demoró mucho más de lo previsto pero los resultados y la posterior reutilización en el IDS compensaron la decisión.

3.3.2. Almacenamiento de características

Las primeras versiones del extractor enviaban la información a la *salida estándar* para comprobar la correcta extracción y estructura de la información. Sin embargo, era necesario implementar un sistema de base de datos para archivar y manejar de forma cómoda la información.

Selección de la base de datos

La primera decisión era si debía utilizar un sistema relacional o no. Personalmente soy un defensor de las bases de datos relacionales y huyo de las nuevas modas que parecen querer usar bases de datos *NoSQL* para todo. Cada tipo de base de datos tiene su aplicación y por sus condiciones de estabilidad e integridad las bases de datos relacionales son mi opción "por defecto". Sin embargo, para el problema que tenía entre manos:

- el esquema iba a ser muy variable, tan variable que ni siquiera lo tenía claro
- primacía de las escrituras sobre las lecturas, bases de datos con muchos registros
- no se necesitaba transaccionalidad ni restricciones ni integridad referencial

Por todos estos motivos me decanté por bases de datos *NoSQL* basada en documentos. La elección de *MongoDB* [29] se debió a:

- ya tenía cierta experiencia previa
- existencia de librería *mgo* [30] y otras posibles opciones
- implementación trivial: mapeo automático de las estructuras en memoria al documento

Nuevamente se realiza un pequeño prototipo para comprobar su viabilidad que inserta, actualiza y lee información desde una base de datos mongo.

Estructura de la base de datos

Una vez seleccionada la base de datos y definida la terminología y codificación, era necesario definir cómo la información iba a estar estructurada en la base de datos.

Una de las decisiones importantes que había que tomar en primer lugar era acerca del etiquetado. ¿Debía etiquetarse una conexión en el momento de la extracción? o por el contrario ¿debería etiquetarse en un proceso posterior?. Por motivos de simplicidad y de una mejor separación de responsabilidades, se toma la decisión de que el extractor no conoce ni sabe de la existencia de malware o no malware, es decir: el extractor no etiqueta.

Tras diversas iteraciones, se determina que la información de un dataset se almacena en las siguientes colecciones de documentos:

- **connections**: almacena información de la conexión, flujos que la conforman, secuencias de handshake, resumen de información transferida por tipo de subprotocolo, etc
- **records**: almacena la información de cada registro tls, su tipo, longitud y timestamp
- **handshakes**: almacena información de handshakes sin cifrar
- **certs**: almacena los certificados vistos en la conversación de manera única

Se utilizan los identificadores de documentos de mongo para relacionar información entre colecciones. Por ejemplo, dentro de un documento *connection* se almacena cada flujo en un documento que a su vez tiene un identificador único. Este identificador único es referenciado en los documentos de las colecciones *records* y *handshakes*. Del mismo modo, los *handshakes* de tipo `Certificates`, contienen referencias a la correspondiente colección *certs* (y que forman parte de los índices de la colección).

Aunque existe duplicidad de información debido a que la colección *handshakes* ya se encuentra integrada como subdocumentos de la colección *connections* (esta colección se agregó en una iteración del desarrollo), se mantuvo la colección *handshakes* por comodidad y eficiencia de algunas consultas. Una vez adaptadas y optimizadas dichas consultas se podría prescindir de dicha colección al ser redundante.

Un ejemplo de registro de tipo records visto en formato JSON

```
{
    "_id" : ObjectId("5ac9104cf24d8f323870afcb"),
    "type" : 22,
    "len" : 55,
    "timestamp" : ISODate("2011-08-15T15:23:24.041Z"),
    "streamId" : ObjectId("5ac9104cbfc7565b76411e6a")
}
```

Decisión: colecciones independientes por dataset

Una posibilidad era crear estas cuatro colecciones y definir un atributo adicional que se agregaría

al índice de la conexión que contuviera el código del dataset. Sin embargo se decidió optar por una solución a priori "menos elegante" (especialmente para los que venimos del mundo relacional) que consistía en crear colecciones para cada dataset de forma independiente con los nombres, por ejemplo, de: `I-CTU-102_connections`, `I-CTU-102_records`, `I-CTU-102_handshakes` y `I-CTU-102_certs`.

El problema evidente de esta decisión es la necesidad de iterar por cada dataset y que no se puede agrupar de manera directa en las consultas de mongo. Sin embargo presenta muchas más ventajas a su favor:

- Existen datasets con millones de registros y su división facilita enormemente el mantenimiento de los datos: borrado, creación, exportación, etc.
- Es más ágil para el gestor a la hora de ejecutar las consultas y es una forma de dividirlas y hacerlas más manejables.
- Nos permite fácilmente seleccionar los dataset empleados para la construcción de modelos.
- Mucho más flexible y escalable en un futuro.

Desde un punto de vista de caja negra, podemos ver la herramienta desarrollada de la siguiente forma:

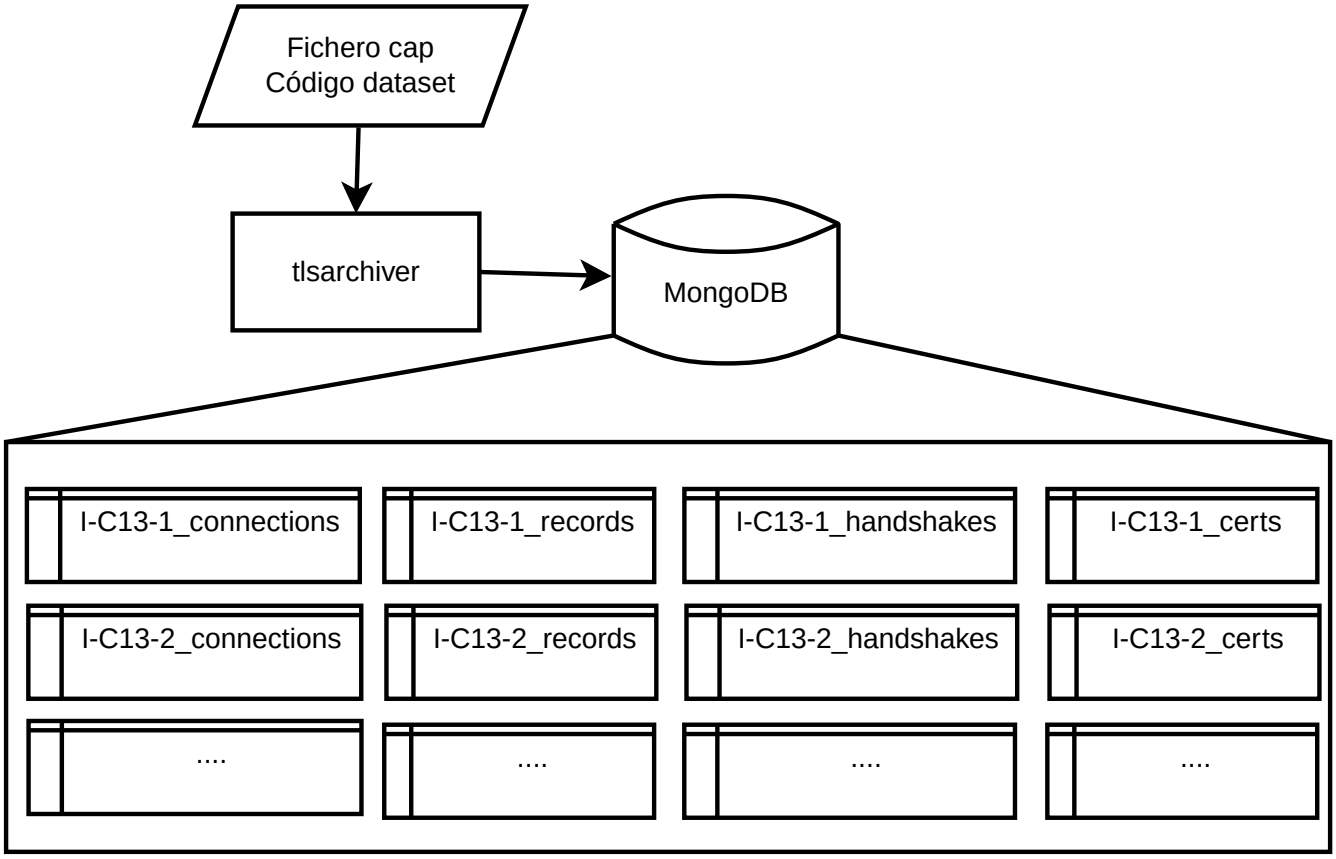

Figura 12. Archivado con `tlsarchiver` en base de datos mongo

3.4. Análisis exploratorio

Tras tener la información referente al tráfico TLS almacenada en base de datos, es necesario realizar un análisis exploratorio sobre los datos. Dicho análisis incluye la navegación por los datos y la utilización de herramientas estadísticas sobre los mismos para obtener conocimiento y validar hipótesis.

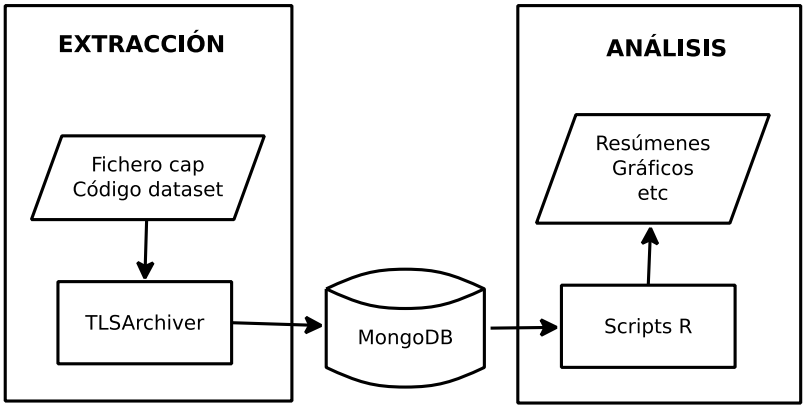

Figura 13. Proceso de análisis exploratorio

3.4.1. Herramientas

Es muy cómodo para el analista poder navegar por la información y lanzar consultas al gestor de bases de datos. Por ello, para la navegación de los datos residentes en la base de datos mongo, opté por una utilidad con interfaz gráfica de usuario llamada *Robo3t* [31].

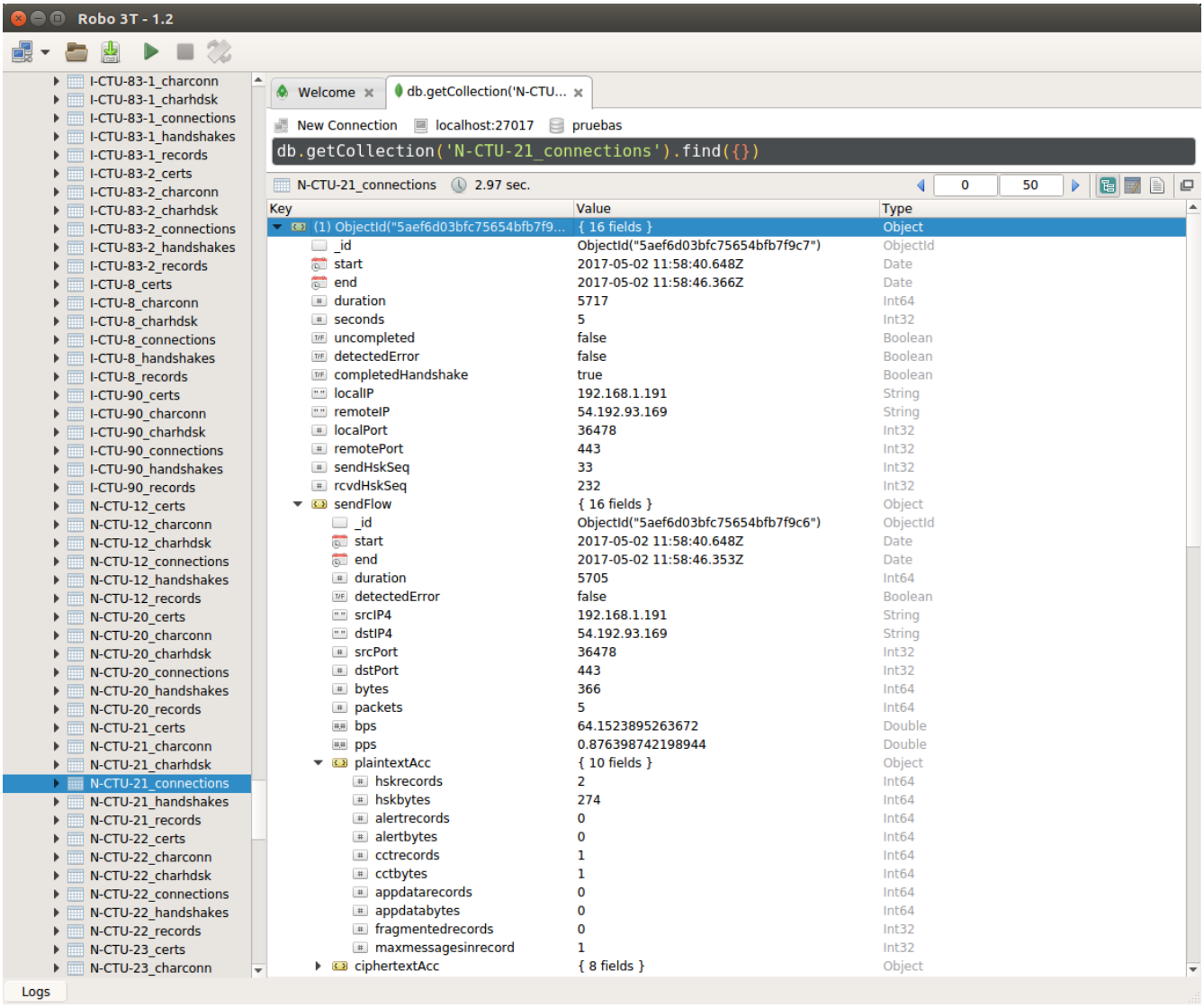

Figura 14. Un documento de tipo `Connection` visto con la utilidad Robo3T

Por otro lado necesitaba herramientas estadísticas con las que preferiblemente se pudiesen crear posteriormente los modelos de ML. En lo que respecta a la herramienta estadística estuve

evaluando el uso de *R* [32] con la interfaz *RStudio* [33] y *Weka* [34]. Finalmente escogí R y RStudio. Los motivos fueron los siguientes en orden de importancia:

- Dispone de numerosas librerías de ML
- Tiene conectores con mongo
- Enorme cantidad de documentación y recursos
- Es posible hacer uso de *OpenCPU* [35] para exponer programas R vía una *API REST*, lo que podría utilizarse en el IDS
- Experiencia previa con R

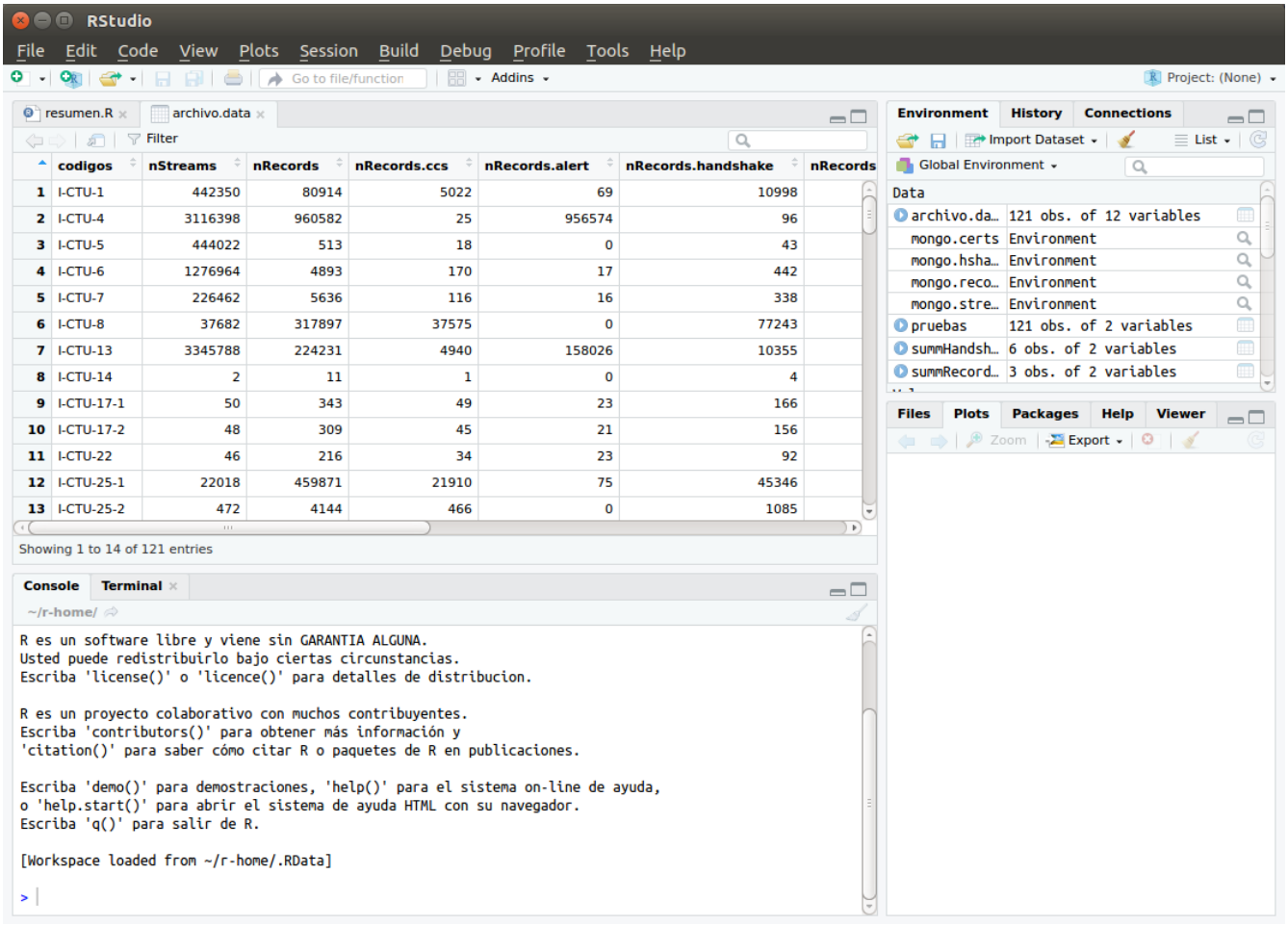

Figura 15. Tratando datos con RStudio

3.4.2. Creación de varios modelos

Mientras realizaba el análisis e iba iterando en el desarrollo del extractor, pude comprobar que numeroso malware no completaba la conexión, enviaba tráfico no TLS, no completaba la secuencia de handshakes, etc. Por ello **existían datos incompletos en conexiones** que no se podrían utilizar o que habría de tratar posteriormente para proporcionárselo al modelo.

También pensé en la funcionalidad desde el punto de vista de tiempo real del extractor:

- Sería ideal que los intentos de conexión de malware se detectasen aunque no pudiese realizarse la conexión.
- Sería ideal que en el mismo momento de producirse el handshake se pudiese evaluar si se trata

de malware para así poder agregar alguna acción como implementar una regla de cortafuegos.

- Sería ideal utilizar la información completa de la conexión tras el cierre ya que se dispondría de mucha más información y el modelo gozaría a priori de mucha mayor precisión.

Por ello decidí que sería necesario implementar al menos tres modelos:

- **Modelo con información del HelloClient**: únicamente utiliza datos del mensaje `HelloClient`.
- **Modelo con información del handshake**: utiliza datos disponibles tras el handshake.
- **Modelo con información del handshake + datos de flujo**: utiliza todos los datos disponibles del handshake junto con los del flujo de datos de la conexión.

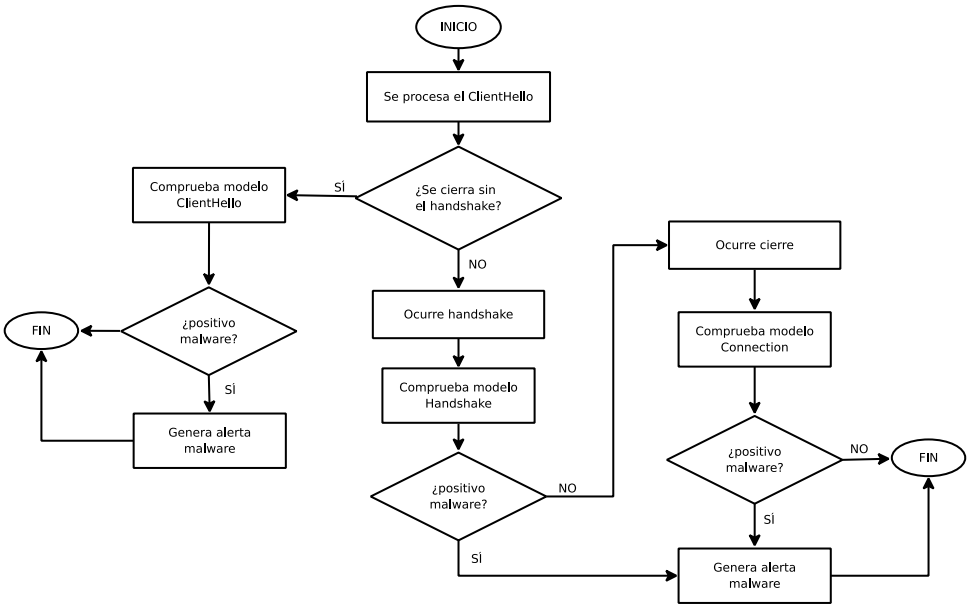

Figura 16. Diagrama de flujo para la aplicación de modelos

A pesar de los modelos propuestos, no se soluciona otro problema de falta de información que existe debido a la información de los certificados. El protocolo TLS permite reanudar la sesión, lo que provoca que no sea necesario el uso de certificados en el handshake y que no se pueda disponer de dicha información en gran cantidad de casos. Estuve valorando la creación de un nuevo modelo que tuviese en cuenta esta información, sin embargo lo descarté por el tiempo disponible y que sería mucho más "elegante" disponer de una caché con las sesiones persistentes para así recuperar la información de dichos certificados. También observé que los resultados observados en otro trabajo similar [36] no parecían excesivamente positivos en lo que respecta a su uso.

3.5. Selección y preparación de los datos

Una vez extraída la información de los ficheros de captura a una base de datos, realizado un análisis exploratorio y definida una serie de agrupaciones de datos que originarían diferentes modelos, era necesario seleccionar qué atributos formarían parte del modelo y procesarlos para poder usarlos con un algoritmo de ML.

El proceso a realizar sería parecido al siguiente:

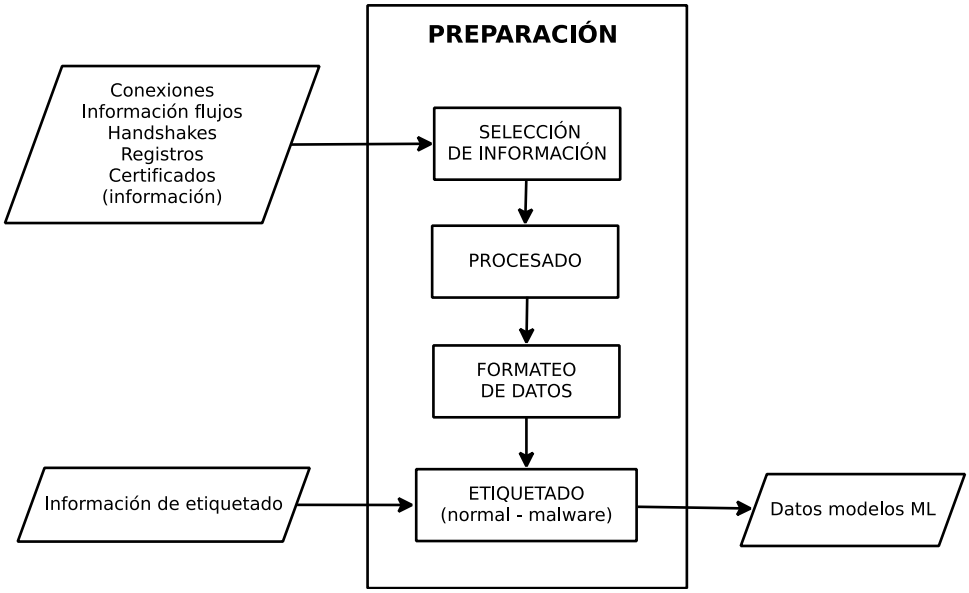

Figura 17. Proceso de preparación de los datos

3.5.1. Selección de atributos

Para la selección de atributos tomé la siguiente decisión: como a priori no son tantos los atributos como para que los modelos y mi equipo sean incapaz de manejarlos, seleccionaré todos los atributos que estén disponibles para cada modelo y dejaré para más adelante la reducción de dimensionalidad, ya sea eliminado de atributos o utilizando técnicas de reducción.

> La optimización prematura es la raíz de todos los males.
>
> — Donald Knuth

A pesar de esto, sí eliminé información que evidentemente no aportaba nada a los modelos como direcciones IPs, puertos y datos de tipo `string` que pudiesen existir (y no pudiesen ser categorizados) como el nombre del servicio en la extensión SNI, empleando en su lugar la longitud de cadena.

3.5.2. Preparación de los datos

En este momento toda la información estaba en un formato cómodo para interpretar por humanos, sin embargo el formato de gran parte de esta información no era útil para su uso directo por parte de los modelos ML.

Conversión de arrays de elementos en categóricos

Esta cuestión se ve fácilmente con un ejemplo, supongamos que dentro de un documento de tipo `Connection`, en el subdocumento `SendFlow`, subdocumento `handshakeSeq`, el primer elemento del array es un handshake de tipo `ClientHello`. Como vemos, ya la propia estructura tiene documentos que puede o no tener y aunque los tuviera no podría pasarlos de manera directa al algoritmo por lo que hay que hacer algo al respecto.

Ahora supongamos que dentro de ese documento tenemos la siguiente información:

```
"extensions" : [
{
    "extcode" : 0,
    "numbytes" : 20
},
{
    "extcode" : 65281,
    "numbytes" : 1
},
{
    "extcode" : 10,
    "numbytes" : 8
},
],
```

Si nos fijamos en la información del array `extensions`, vemos que tiene subdocumentos con un código de extensión y con el número de bytes usados. Para que esto lo pudiéramos utilizar deberíamos convertirlo en algo parecido a esto otro:

```
"e_0" : TRUE, "e_0_len" : 20, "e_65281": TRUE, "e_65281_len" : 1,
"e_10": TRUE, "e_10_len" : 8
```

Hemos convertido la información en categórica (realmente lógica) y numérica, sin embargo existen muchas más extensiones, por lo tanto tendremos que crear una nueva variable para todas las extensiones posibles de TLS e indicar los valores `FALSE` y `0`:

```
"e_0" : TRUE, "e_0_len" : 20, "e_1" : FALSE, "e_1_len : 0,
"e_10" : TRUE, "e_10_len" : 8....
```

Las tareas de conocer todas las extensiones posibles, algoritmos de firma posibles, versiones tls posibles, etc y todas las transformaciones se implementan en el producto desarrollado junto a este trabajo `r-tls`.

Secuencia de handshake

Otro problema era cómo modelar una secuencia de handshakes, estas características eran importantes porque al fin y al cabo, unas secuencias podrían ser incorrectas o deberían mostrar congruencia con el resto de elementos (por ejemplo una secuencia de recuperación sin identificadores de sesión no tienen sentido). Por ello utilicé un sencillo algoritmo para generar un valor de secuencia que sería independiente para el envío y la recepción.

```
valor_secuencia = 0
posicion = 1
PARA handshake EN secuencia
    valor_secuencia = valor_secuencia + (posicion * handshake.tipo)
    posicion = posicion + 1
FIN PARA
```

Como los posibles valores resultantes son limitados, hago esta variable como si fuese categórica y la proceso como tal.

Atributos calculados

Existen atributos que pueden aportar eficiencia a los modelos que son calculados a partir de otros atributos. Un ejemplo claro son los `bytes por segundo`. Se tomó la decisión de que estos atributos se calculasen por el extractor tras el cierre de conexión y que se almacenasen en la base de datos para así simplificar mucho el posterior análisis y la preparación de los modelos. A pesar de esto, como se verá posteriormente, podrían implementarse nuevos atributos de este tipo en nuestra *pipeline* de procesamiento.

3.5.3. Implementación del proceso

El primer intento para seleccionar y preparar los datos fue tratar de obtener todos los datos desde la base de datos directamente a R para manipularlos y prepararlos íntegramente con esta herramienta. Sin embargo, la práctica demostró que la carga desde MongoDB de un número tan elevado de documentos de esa complejidad y su posterior manipulación en R **era muy ineficiente** en términos de memoria y CPU.

Por estos motivos de ineficiencia, se optó por un enfoque distinto en el que se empleó el propio *Framework de agregación de MongoDB* [37] para realizar todo el proceso de selección y transformación del formato de los datos.

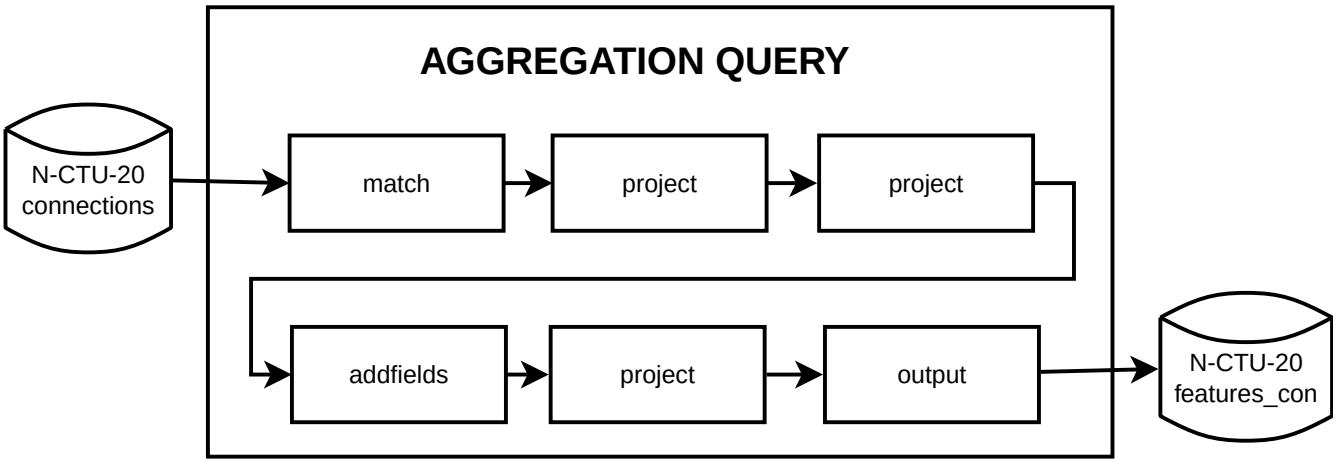

Figura 18. Query de agregación que construye una colección con las características

Como se puede ver en la imagen, la query coge los datos de la colección y realiza una serie de acciones. El `match` hace un filtrado, los `project` seleccionan y renombran campos, y el `addfields` agrega nuevos campos usando operaciones disponibles en mongo (parecido a un lenguaje de programación). Finalmente el `output` hace que almacene la salida en una nueva colección.

Esta parte del proceso es la siguiente:

1. se generan dinámicamente las querys de agregación en lenguaje R y se ejecutan en MongoDB desde el mismo runtime de R usando para ello la bilioteca *mongolite*[38]
2. estas querys definen una pipeline para el framework de agregación que da lugar a nuevas colecciones con documentos planos con la información de factores correctamente definida, etc.

Al final del proceso tenemos tres colecciones diferentes nuevas por cada dataset:

- **features_ch**: características de ClientHello
- **features_hsk**: características de Handshake
- **features_con**: características de Connection

3.5.4. Etiquetado de los datos

Durante la carga de datos desde las colecciones de características ya preparados, simplemente se etiquetan agregando una columna en R. Como ya dije, se prescinde de los `Mixed` por la complejidad (y casi imposibilidad) de etiquetarlos correctamente y únicamente se etiquetan los dataset con `Normal` o `Infected`.

Nótese que para el etiquetado estamos asumiendo que todas las conexiones que aparecen en los dataset de tipo malware son de este tipo. Si existiesen conexiones legítimas como pudiesen ser procesos de actualización en segundo plano en dichas capturas, también serán consideradas como malware.

3.5.5. Datasets finales

Un nuevo modelo Fingerprint

En una nueva iteración y a la vista de los resultados obtenidos para los distintos modelos, se crea un nuevo modelo subconjunto del modelo ClientHello llamado **Fingerprint**. En este modelo lo que se hace es eliminar los atributos de longitud de extensiones y hacer una operación de unicidad para que queden solamente los elementos únicos de cada dataset. Dicho de otra manera: para que queden únicamente los distintos fingerprints existentes en cada dataset. Posteriormente se aplica de nuevo la unicidad por clase. De este modo se busca probar la efectividad del fingerprinting como mecanismo de clasificación.

Composición final

El número de características de los distintos modelos queda como sigue, hay que tener en cuanta que la inmensa mayoría son de tipo lógico `TRUE` o `FALSE`

Table 4. Número de características de los modelos

Modelo	Características
Fingerprint	231
ClientHello	258
Handshake	498
Connection	519

Para la composición de muestras final, debido a la existencia de un mayor número de datasets de tipo infected y a que algunos de ellos tenían hasta cientos de miles de conexiones, se hizo lo siguiente:

1. Si dataset es `Normal` entonces incorporar todo
2. Si dataset es `Infected` entonces incorporar un máximo de 5000

De este modo llegamos a los siguiente conjuntos finales de muestras etiquetadas:

Table 5. Muestras disponibles para la construcción y prueba de los modelos

Modelo	Infected	Normal
Fingerprint	443	53
ClientHello	142619	73439
Handshake	139757	71739
Connection	139757	71739

3.6. Construcción de los modelos

Como vimos, ya tenemos los datos preparados y etiquetados para usar con los diferentes algoritmos. Ahora tendremos que decidir qué algoritmos seleccionaremos, cómo los usaremos y cómo crearemos y validaremos su eficacia. Finalmente seleccionaremos el que será implementado en nuestro IDS.

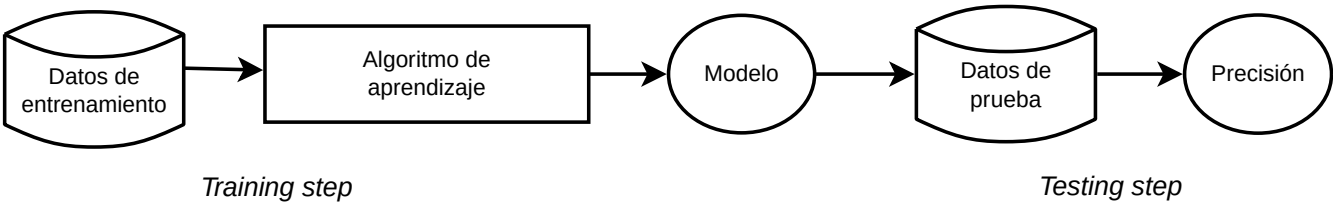

Figura 19. Proceso de creación y validación de un modelo basado en aprendizaje supervisado, fuente [6]

3.6.1. Selección de algoritmos y uso

Los criterios que utilicé para la selección, dada mi nula experiencia y conocimiento de ML, fueron básicamente los siguientes:

1. Que tuvieran ya implementación en R.
2. Que fuesen fáciles de usar.

3. Que fuesen los más básicos o conocidos.
4. Que aportasen valor explicativo.

Debido a esto seleccioné los tres algoritmos "básicos":

- **Decission tree**: aporta gran valor explicativo observando el árbol de decisión generado.
- **Naïve Bayes**: aporta gran valor explicativo observando las probabilidades a priori del modelo generado.
- **SVM**: aparentemente tenía muy buena capacidad predictiva.

Posteriormente agregué un cuarto algoritmo **Random Forest** debido a que usarlo era trivial y los resultados prometían ser mejores que los obtenidos con los otros modelos.

La creación en R de modelos es muy sencilla, básicamente hay que utilizar la librería adecuada de algoritmo y utilizar la función correspondiente.

Extracto de código en R que crea un modelo Naïve Bayes

```
print("Creating model...")
start_time <- Sys.time()
model <- naiveBayes(classification ~ ., data = features.df[training.ids,])
end_time <- Sys.time()
consumed_time <- difftime(end_time, start_time)
print(consumed_time)
```

3.6.2. Creación y validación de los modelos

Para crear y validar los modelos dividiremos el dataset que obtuvimos en el apartado de preparación en dos partes: entrenamiento y prueba. Para ello, utilizando bibliotecas de R, parto el dataset en `80%` de entrenamiento y un `20%` de prueba.

Quedando los datasets de la siguiente forma:

Table 6. División del dataset en entrenamiento y prueba

Modelo	Entrenamiento (I)	Entrenamiento (N)	Prueba (I)	Prueba (N)
Fingerprint	335	43	88	10
Clienthello	114096	58752	28523	14687
Handshake	111804	57069	27950	14267
Connection	111804	57069	27950	14267

En un principio tenía previsto también realizar la técnica de validación cruzada (o *cross validation*), sin embargo no la implementé por cuestiones de tiempo y recursos: el número de tipos de modelo y el número de algoritmos ya era bastante elevado y no disponía de una máquina independiente en la que ir procesando (en caso de 10 iteraciones implicaría multiplicar por 10 todo el proceso, ya de por si bastante costoso como se verá en Resultados).

En el capítulo dedicado a Resultados se muestran los resultados obtenidos por los distintos modelos. Son de especial valor las matrices de confusión obtenidas de la aplicación de los algoritmos. A partir de dichos valores se pueden calcular una serie de métricas de clasificación.

De estas métricas, es especialmente importante el *FPR* (*False Positive Rate*), que expresa una probabilidad de que conexiones normales sean etiquetadas como malware. Esto es muy importante: si un IDS da muchos falsos positivos deja de ser útil. Otra métrica importante es la precisión indicada como *ACC* (*Accuracy*), que indica qué precisión tiene el clasificador.

Table 7. Valores de precisión (ACC) y falsos positivos (FPR)

Modelo	**Decission tree**	**Naïve Bayes**	**SVM**	**Random Forest**
Fingerprint	0,9795918367	0,8367346939	0,9285714286	0,9795918367
	0,2	0,1	0,2	0,2
ClientHello	0,9787086323	0,9555426984	0,9843323305	0,9827354779
	0,0125961735	0,0558316879	0,0010893988	0,0008851365
Handshake	0,9783262667	0,9321363432	0,9914015681	0,9861903972
	0,0119857013	0,0585266699	0,0022429382	0,0016121119
Connection	0,9783262667	0,8942132316	0,7166781155	0,9919226852
	0,0119857013	0,0422653676	0,8367561506	0,0014018364

Hay que tener en cuenta que el modelo Fingerprint se ha realizado sobre un conjunto muchísimo más reducido de datos, por lo que hay que tomar los datos con mucha precaución. Dicho esto, vemos que la precisión es muy buena (especialmente en Decission tree) lo que nos hace indicar que este modelo clasifica bastante bien. Si entramos un poco más en detalle, vemos que existen instancias con los mismos atributos en las dos clases. Concretamente `13` sobre un total de `496` lo que hace un `0,2%` de fingerprints coincidentes de ambas clases. Sin embargo los falsos positivos son muy elevados con casi el `20%`. Esto lo descartaría como clasificador pero nos sirve para comprobar que los datos del fingerprint en nuestro dataset prácticamente definen si son malware o no y pueden ser de utilidad para clasificadores intermedios (como se propuso en la fase de análisis). Si observamos el árbol de decisión creado, es muy clarificador ver cómo el malware cumple algunas hipótesis que lanzamos en lo que respecta a usos de algoritmos y firma más débiles. Fijémonos cómo el uso del algoritmo de firma `2052` ya determina que el tráfico es normal.

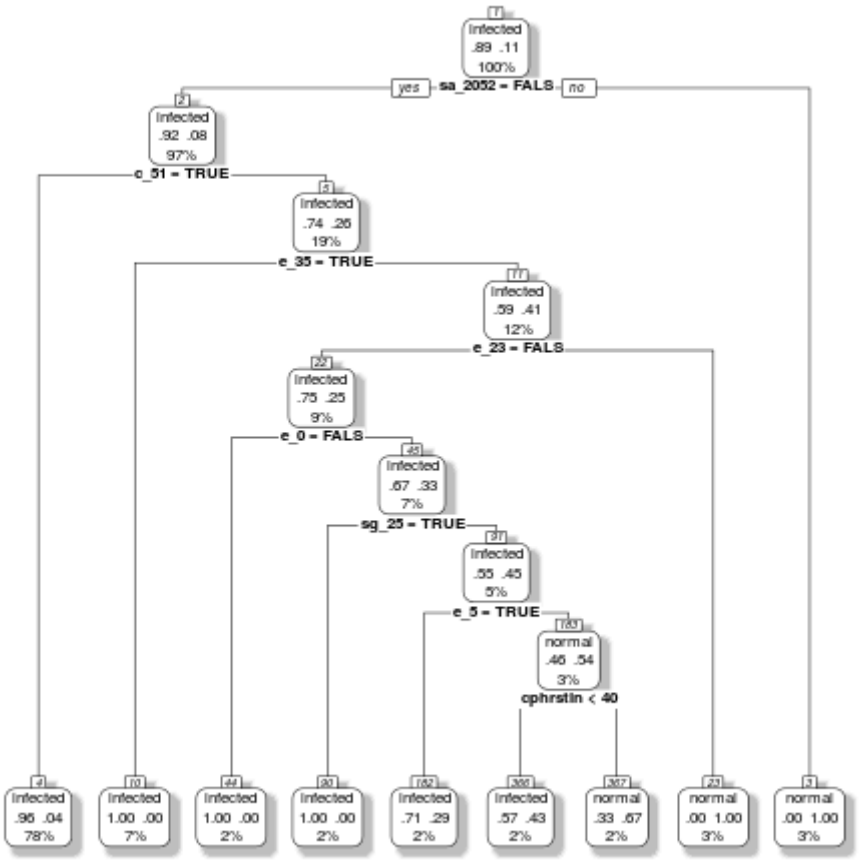

Figura 20. Árbol de decisión creado para el modelo fingerprint

El resto de modelos tienen un volumen de muestras similares, lo que los hace en cierta forma comparables. Como vemos, la precisión va aumentando en todos los modelos (excepto SVM que parece que dio error creando el Connection) y se reducen los falsos positivos. Esto parece confirmar que el incremento de atributos mejora la capacidad de predicción del modelo. Sin embargo, es de esperar que no todos los atributos lo hagan, por lo que el siguiente paso sería eliminar atributos y/o tratar de resumirlos.

Vemos que el mejor modelo es el Connection con el algoritmo Random Forest, que tiene una precisión de **99%** y falsos positivos **0,1%**.

Debemos tomar estos datos con mucha precaución ya que estos datos no representan al Universo de todos los posibles malware y software benigno que hagan uso de TLS. Lo que quiere decir es que tenemos un clasificador muy bueno **para conexiones no vistas del software empleado para el entrenamiento**. Si es capaz de generalizar tan bien para cualquier software lo vamos a ver enseguida.

3.6.3. Selección de algoritmo

Para seleccionar nuestro algoritmo deberíamos tener en cuenta lo siguiente:

1. Tiempo de clasificación viable para un solución de este tipo (orden de milisegundos)
2. Buenos valores de FPR y ACC, especialmente falsos positivos.

3. Fácil uso y despliegue.
4. Deseable que tiempo de entrenamiento sea reducido.

Aunque los tiempos de clasificación de Random Forest son superiores a Decission Tree, la precisión del primero y falsos positivos son mucho mejores. Además vemos que los tiempos de clasificación son aceptables. Si tomamos el mayor tiempo de clasificación y dividimos por el número de muestras clasificadas estamos dentro de requerimientos: `5,178938 / (27950+14267) = 0,000122674 s`

No es lo mismo clasificar `1000` muestras que clasificar `1`, las operaciones se realizan de forma vectorial en R, por lo que el tiempo final puede llegar a ser el mismo. Por ello es muy importante agrupar las muestras para que el clasificador tenga un rendimiento aceptable.

El aspecto negativo de Random Forest es el tiempo de construcción de los modelos, que en el peor de los casos son unos `25` minutos. Se considera que esto es asumible.

Por los motivos enumerados, y debido a que por simplificación y gestión de dependencias sólo escogeremos un algorimo, nos quedaremos con Random Forest y sus modelos `ClientHello`, `Handshake` y `Connection`.

3.7. Desarrollo del IDS

Ya tenemos los modelos de ML listos para ser usados, pero para ello tenemos que integrarlos en una herramienta que sea capaz de usarlos con conexiones en "tiempo real".

3.7.1. Separación entre motores de ML e IDS

Cuando estudié el uso de R, descubrí la existencia de la aplicación OpenCPU que permite exponer aplicaciones desarrolladas en R a través de una API REST. Esto hace posible dividir las tareas de análisis de datos y creación de modelos mediante el lenguaje R y la implementación de herramientas en cualquier otro lenguaje que hagan uso de ellas, lo que era ideal para este proyecto. Básicamente tenemos por un lado el sistema IDS y por otro lado el "cerebro" al cual consultaremos por la probabilidad de existencia de malware:

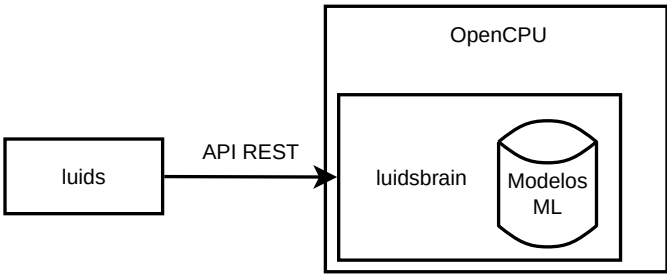

Figura 21. Uso de la API REST del sistema luIDS

Nótese además que esta solución es escalable y haría posible la implementación de un servicio cloud.

3.7.2. Creación de la solución IDS

La solución desarrollada podría verse como tres grandes bloques:

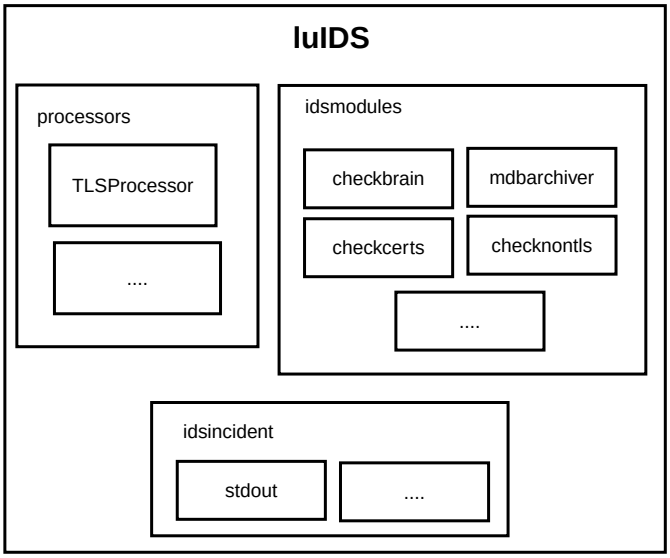

Figura 22. Diagrama de bloques luIDS

- **Procesamiento**: escucha en un puerto de red y procesa el tráfico, transformándolo a estructuras de datos manejables por los módulos.
- **Módulos**: están pensados principalmente para las tareas de detección, pero pueden usarse para más cosas. Por ejemplo, el módulo `checkbrain` es el encargado de solicitar a la API REST una probabilidad de infección y el módulo `mdbarchiver` que almacena la información procesada en mongodb (con este módulo se implementa el extractor).
- **Gestión de incidentes**: se implementarían componentes para la gestión de incidentes generadas por los módulos IDS. Se podrían implementar diferentes estrategias, pero para la prueba de concepto simplemente se ha implementado un manejador básico que consiste en escribir por pantalla la incidencia.

La integración de módulos IDS con la capa de procesamiento se realiza mediante **un mecanismo de hooks** que permite fácilmente agregar la lógica que se desee: desde hacer un archivado, comprobar una heurística, llamar a un webservice, etc.

Algunos de los hooks definidos en la pipeline son:

- `newConnection`: se produce cuando el primer flujo de una nueva conexión llega.
- `beginStream`: se produce con el primer byte procesado de un stream.
- `afterRecord`: se produce tras el reensamblado de un tlsrecord.
- `afterCloseStream`: se produce tras el cierre de uno de los flujos.
- `afterCloseConnection`: se produce tras el cierre de los dos flujos de una conexión.
- `beginEncription`: se produce cuando se recibe el mensaje CCT en ambos flujos y se da por completado el handshake (al menos de cara a nosotros).

Como puede verse, se trata de una solución modular desarrollada en distintos paquetes. Gracias a esto se pueden generar los siguientes binarios especializados que conforman el software luIDS:

- **tlsarchiver**: extrae de ficheros pcap las características y las almacena en una base de datos mongo.
- **tlscheckmalware**: comprueba la existencia de malware en una captura.
- **luids**: comprueba la existencia de malware en tiempo real sobre una interfaz de red.

3.8. Despliegue de la solución

3.8.1. Escenario de despliegue

Para desplegar la solución en real, creo un escenario con dos máquinas virtuales:

- *luids*: máquina con el sistema completo instalado junto con OpenCPU, tiene dos interfaces una con el host y otra a la red de pruebas, haciendo nat a todas las máquinas de la red de pruebas dando salida a internet a través de la red interna. Además se instala un cortafuegos que evita el tráfico desde la red de pruebas hacia la red interna, lo que evita que el malware trate de explorarla.
- *maquina1*: máquina con sistema windows 7.

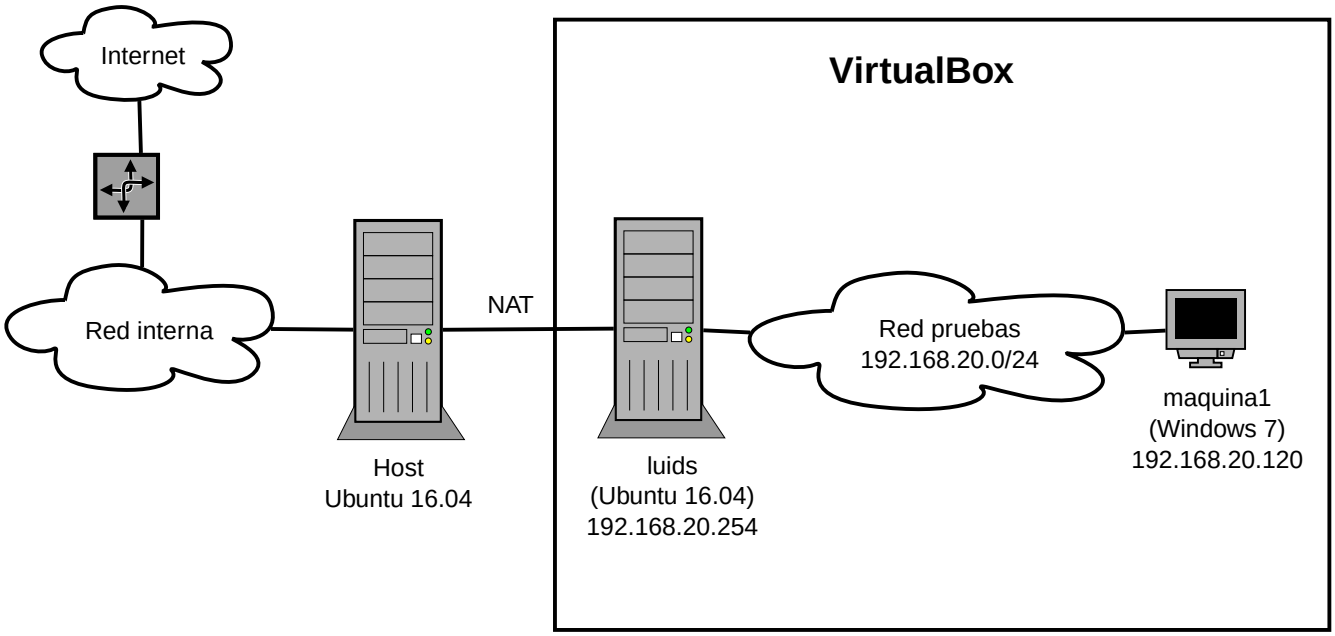

Figura 23. Escenario de pruebas

3.8.2. Resultados

Antes de realizar la prueba sobre el entorno virtual, realizo una prueba en la máquina Host, es decir en mi máquina de desarrollo. Compruebo que el sistema funciona correctamente navegando con Firefox y Chrome y no da ningún tipo de aviso. Esto es una buena noticia, ya que las versiones de mis navegadores deben de ser más modernas que las de las capturas y por lo tanto los ha generalizado correctamente.

Sin embargo aparece el primer problema: detecta a Drobpox como posible malware. Esto en cierto modo podría llegar a interpretarse también como un buen resultado, ya que tiene lógica que una aplicación como Dropbox pudiera ser detectada como malware si no se ha empleado como

conjunto de entrenamiento al sistema.

Posteriormente realizo la prueba sobre el entorno virtual y enciendo la máquina Windows 7. Al momento de encenderla, detecta al actualizador Windows Update y a Internet Explorer 9 como posible malware. Esto podría deberse a varios factores:

1. no hay en el conjunto de entrenamiento de la clase normal sistemas windows 7 y los handshakes que realizan tienen más similitud con las muestras de malware usadas (parece que los sistemas fueron Windows 10, Ubuntu y Kali),
2. al tratarse de un navegador antiguo no emplea extensiones y suites de cifrado modernas y los modelos lo han generalizado a la clase malware,
3. hay malware dentro del conjunto de entrenamiento que imita a la perfección el comportamiento de windows update y explorer (o usa las mismas librerías),
4. el conjunto de entrenamiento de la clase malware contiene muestras de windows update y/o navegador explorer

Cuando usé los datasets era consciente de la posibilidad de existencia de tráfico "bueno" dentro del conjunto de entrenamiento. La descarté debido a la existencia de numerosos datasets etiquetados como malware de días de duración y que no tenían ningún tipo de tráfico TLS. Por ello deduje que si no existían era porque se habían desactivado las actualizaciones automáticas en los equipos de captura y podía asumir que todo el tráfico era del malware.

Llegado a este punto, creo los datasets `N-LUI-001` y `N-LUI-002` con capturas de Windows 7 actualizándose y navegando con Internet Explorer 9 y 11 y otra de mi host con Dropbox. Después de esto, vuelvo a generar todos los modelos y despliego nuevamente la solución. Tras esta actualización de modelo, la solución ya no detecta ningún tráfico como malware.

A continuación, realizo una instantánea de seguridad de la máquina virtual del Windows 7 y procedo a instalar un Botnet. Tras unos segundos compruebo que en la consola avisa de posibles conexiones de malware, lo que indica que está identificando el malware de forma correcta.

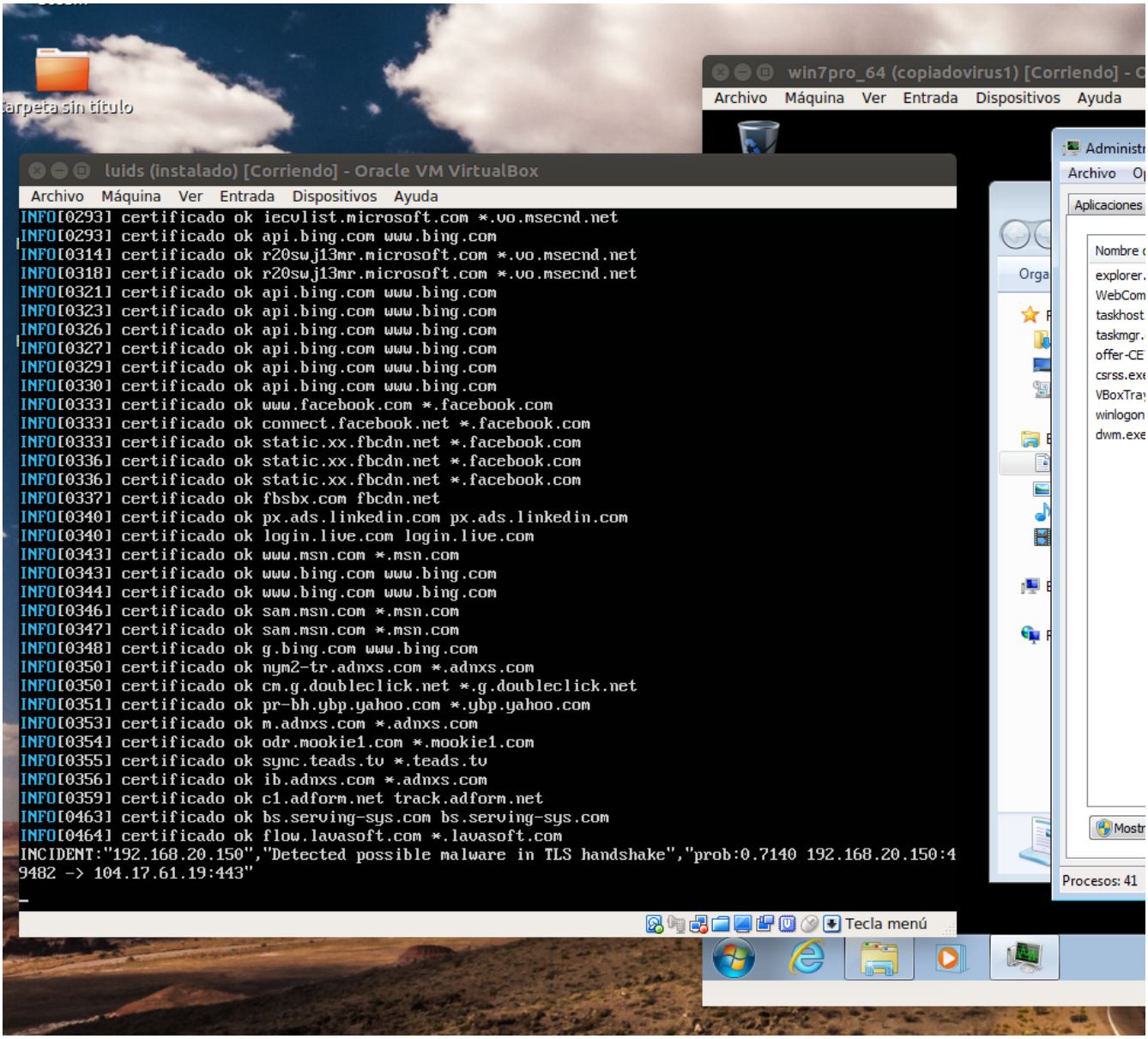

Figura 24. luIDS en ejecución detectando un malware

Chapter 4. Resultados

Este capítulo es una recopilación de los resultados obtenidos.

4.1. Carga de datasets

Table 8. Composición de datasets para los modelos, atributos y tiempos de carga

Modelo	Atributos	Infected	Normal	Tiempo
Fingerprint	231	443	53	1.214629 hours
ClientHello	258	142619	73439	6.853552 mins
Handshake	498	139757	71739	23.501 mins
Connection	519	139757	71739	19.47236 mins

Table 9. Muestras usados para entrenamiento

Modelo	Entrenamiento (I)	Entrenamiento (N)	Prueba (I)	Prueba (N)
Fingerprint	335	43	88	10
Clienthello	114096	58752	28523	14687
Handshake	111804	57069	27950	14267
Connection	111804	57069	27950	14267

4.2. Creación de modelos

Table 10. Tiempos de creación de modelos

Modelo	Decission Tree	Naïve Bayes	SVM	Random Forest
fingerprint	0.9987288 secs	0.2278409 secs	1.25341 secs	1.704664 secs
clienthello	52.17823 secs	3.599784 secs	1.02626 hours	8.637485 mins
handshake	1.48416 mins	6.264904 secs	1.923556 hours	21.3917 mins
connection	1.602666 mins	6.525112 secs	20.74323 hours	25.34188 mins

4.3. Prueba de modelos

Table 11. Tiempos de prueba de modelos (de todo el training set)

Modelo	Decission Tree	Naïve Bayes	SVM	Random Forest
fingerprint	0.2789588 secs	2.180001 secs	0.2564373 secs	0.2245011 secs
clienthello	2.243457 secs	5.326148 mins	1.333376 mins	3.070031 secs
handshake	4.16083 secs	10.24877 mins	1.746116 mins	4.942674 secs
connection	4.255155 secs	10.6337 mins	46.23784 mins	5.178938 secs

Matrices de confusión modelos Fingerprint

```
     Decission Tree                   Naïve Bayes
                Predicho                         Predicho
Actual     infected normal    Actual     infected normal
  infected       88      0      infected       73     15
  normal          2      0      normal          1      9

           SVM                        Random Forest
                Predicho                         Predicho
Actual     infected normal    Actual     infected normal
  infected       83      5      infected       88      0
  normal          2      8      normal          2      0
```

Table 12. Métricas de clasificación modelos Fingerprint

Métrica	**Decission tree**	**Naïve Bayes**	**SVM**	**Random Forest**
ERR	0,0204081633	0,1632653061	0,0714285714	0,0204081633
ACC	0,9795918367	0,8367346939	0,9285714286	0,9795918367
TPR	1	0,8295454545	0,9431818182	1
FPR	0,2	0,1	0,2	0,2
PRE	0,9777777778	0,9864864865	0,9764705882	0,9777777778
F1	0,9887640449	0,9012345679	0,9595375723	0,9887640449

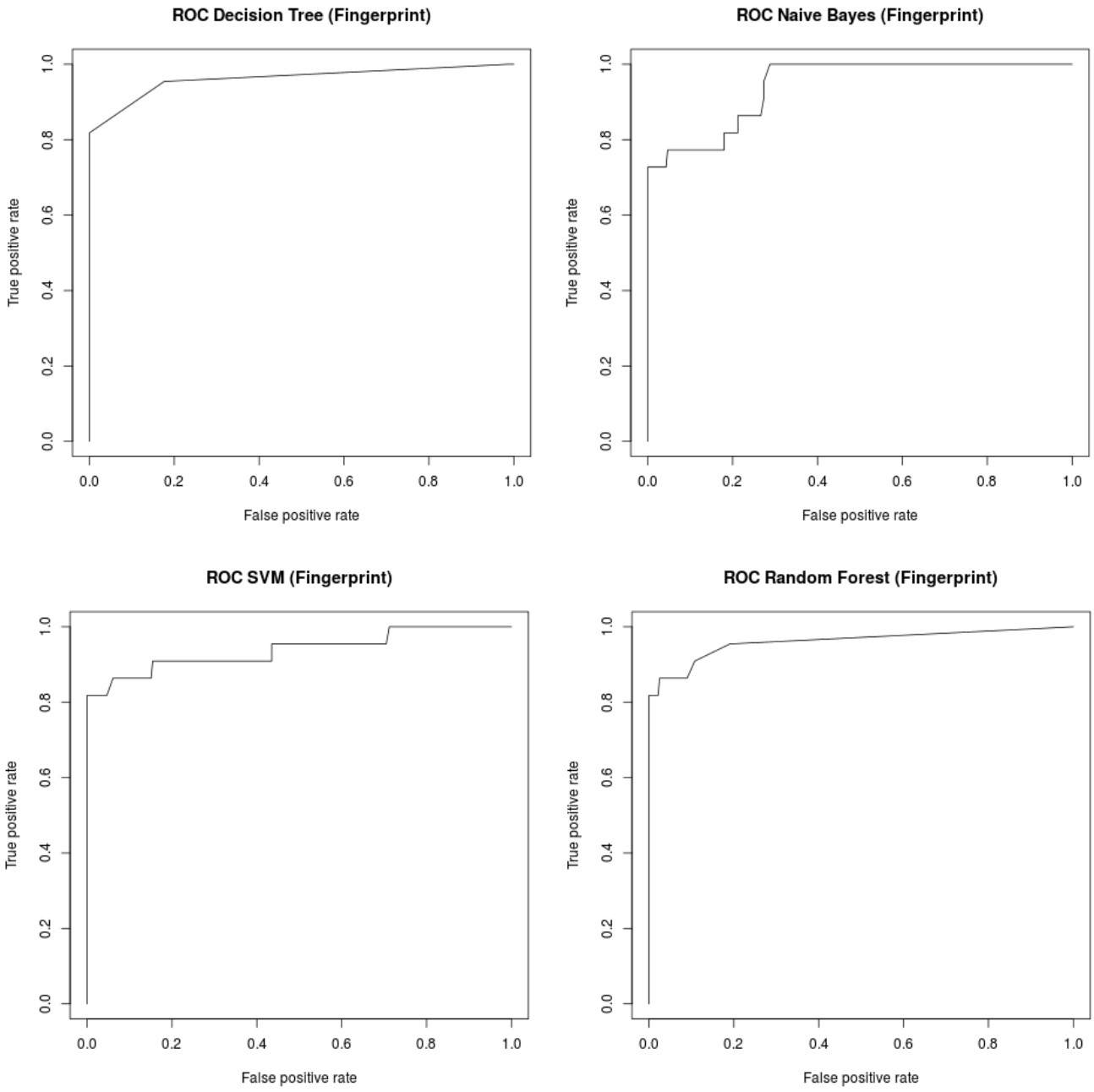

Figura 25. Curvas ROC modelos Fingerprint

Matrices de confusión modelos ClientHello

```
     Decission Tree                  Naïve Bayes
                Predicho                        Predicho
Actual     infected normal    Actual     infected normal
  infected    27788    735      infected    27422   1101
  normal        185  14502      normal        820  13867

           SVM                       Random Forest
                Predicho                        Predicho
Actual     infected normal    Actual     infected normal
  infected    27862    661      infected    27790    733
  normal         16  14671      normal         13  14674
```

Table 13. Métricas de clasificación modelos ClientHello

Métrica	Decission tree	Naïve Bayes	SVM	Random Forest
ERR	0,0212913677	0,0444573016	0,0156676695	0,0172645221
ACC	0,9787086323	0,9555426984	0,9843323305	0,9827354779
TPR	0,9742313221	0,9613995723	0,9768257196	0,9743014409
FPR	0,0125961735	0,0558316879	0,0010893988	0,0008851365
PRE	0,9933864798	0,9709652291	0,9994260707	0,9995324246
F1	0,9837156613	0,9661587246	0,9879966667	0,9867556723

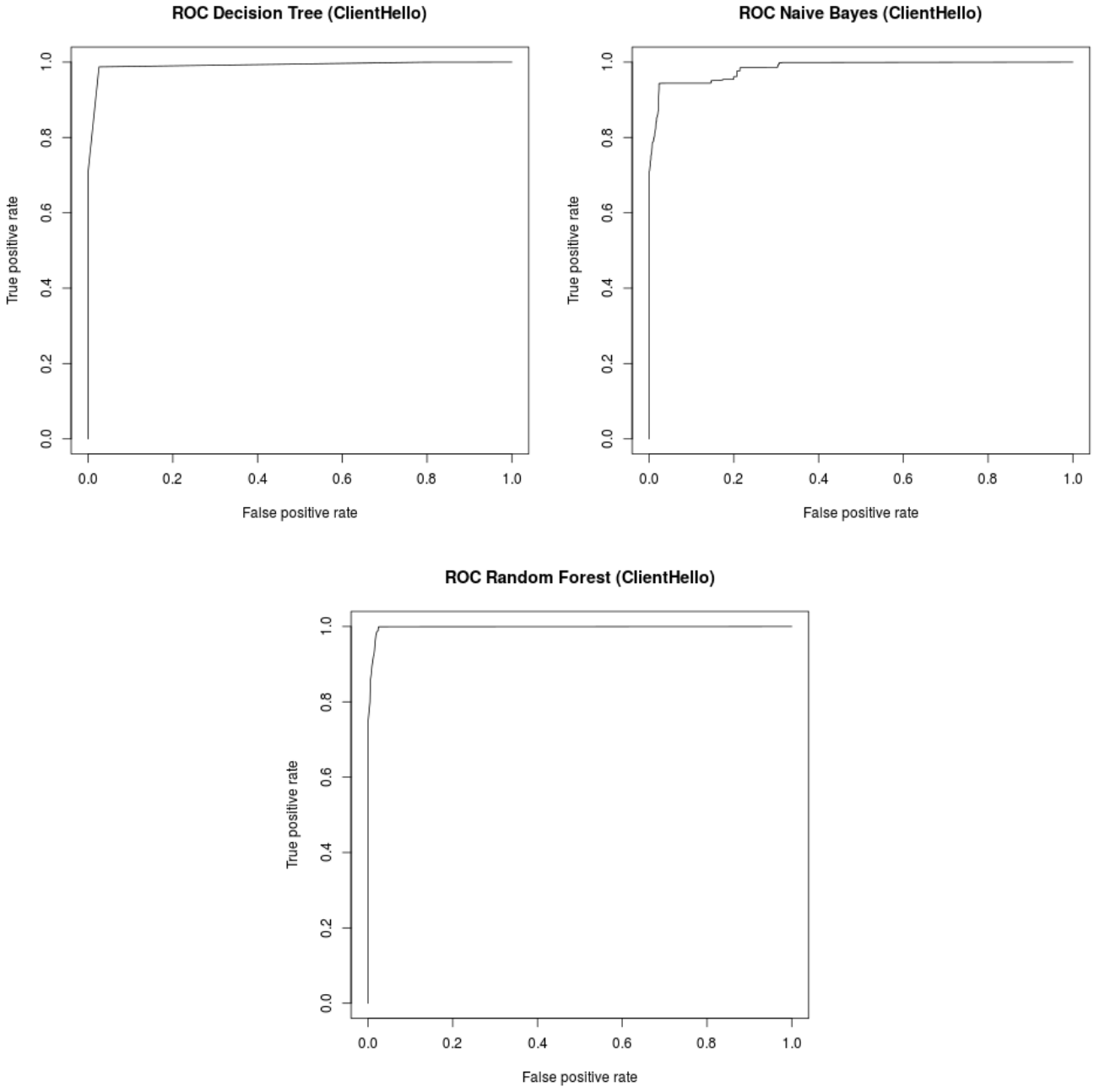

Figura 26. Curvas ROC modelos ClientHello

Matrices de confusión modelos Handshake

```
     Decission Tree                   Naïve Bayes
                Predicho                         Predicho
Actual     infected normal    Actual     infected normal
  infected    27206    744      infected    25920   2030
  normal        171  14096      normal        835  13432

           SVM                        Random Forest
                Predicho                         Predicho
Actual     infected normal    Actual     infected normal
  infected    27619    331      infected    27390    560
  normal         32  14235      normal         23  14244
```

Table 14. Métricas de clasificación modelos Handshake

Métrica	Decission tree	Naïve Bayes	SVM	Random Forest
ERR	0,0216737333	0,0678636568	0,0085984319	0,0138096028
ACC	0,9783262667	0,9321363432	0,9914015681	0,9861903972
TPR	0,9733810376	0,9273703041	0,988157424	0,9799642218
FPR	0,0119857013	0,0585266699	0,0022429382	0,0016121119
PRE	0,993753881	0,9687908802	0,9988427182	0,999160982
F1	0,9834619625	0,9476281876	0,9934713404	0,9894695013

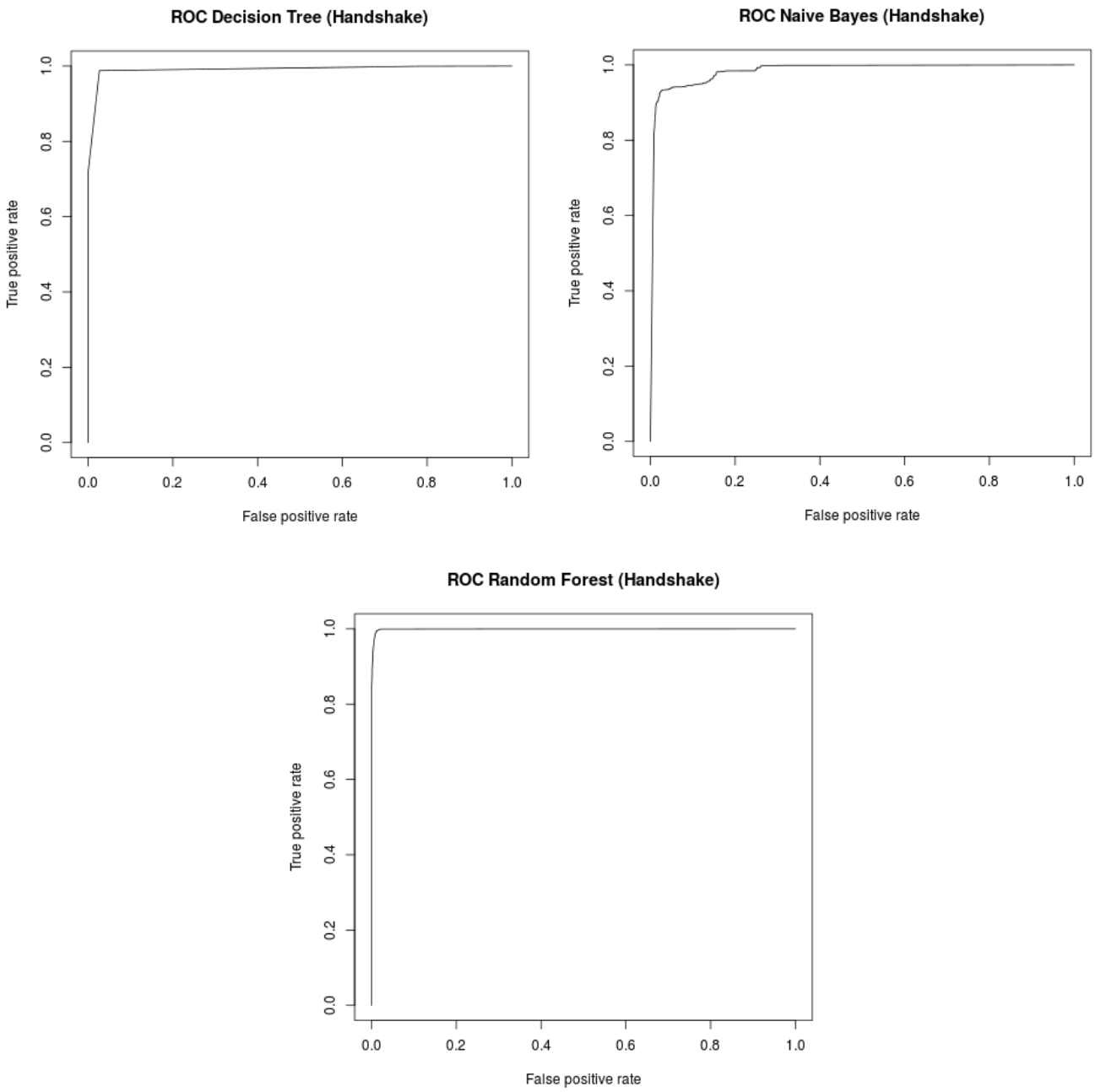

Figura 27. Curvas ROC modelos Handshake

Matrices de confusión modelos Connection

```
     Decission Tree                  Naïve Bayes
                Predicho                        Predicho
Actual     infected normal    Actual     infected normal
  infected    27206    744      infected    24087   3863
  normal        171  14096      normal        603  13664

           SVM                        Random Forest
                Predicho                        Predicho
Actual     infected normal    Actual     infected normal
  infected    27927     23      infected    27629    321
  normal      11938   2329      normal         20  14247
```

Table 15. Métricas de clasificación modelos Connection

Métrica	Decission tree	Naïve Bayes	SVM	Random Forest
ERR	0,0216737333	0,1057867684	0,2833218845	0,0080773148
ACC	0,9783262667	0,8942132316	0,7166781155	0,9919226852
TPR	0,9733810376	0,8617889088	0,999177102	0,9885152057
FPR	0,0119857013	0,0422653676	0,8367561506	0,0014018364
PRE	0,993753881	0,9755771567	0,7005393202	0,9992766465
F1	0,9834619625	0,9151595745	0,8236230922	0,9938667962

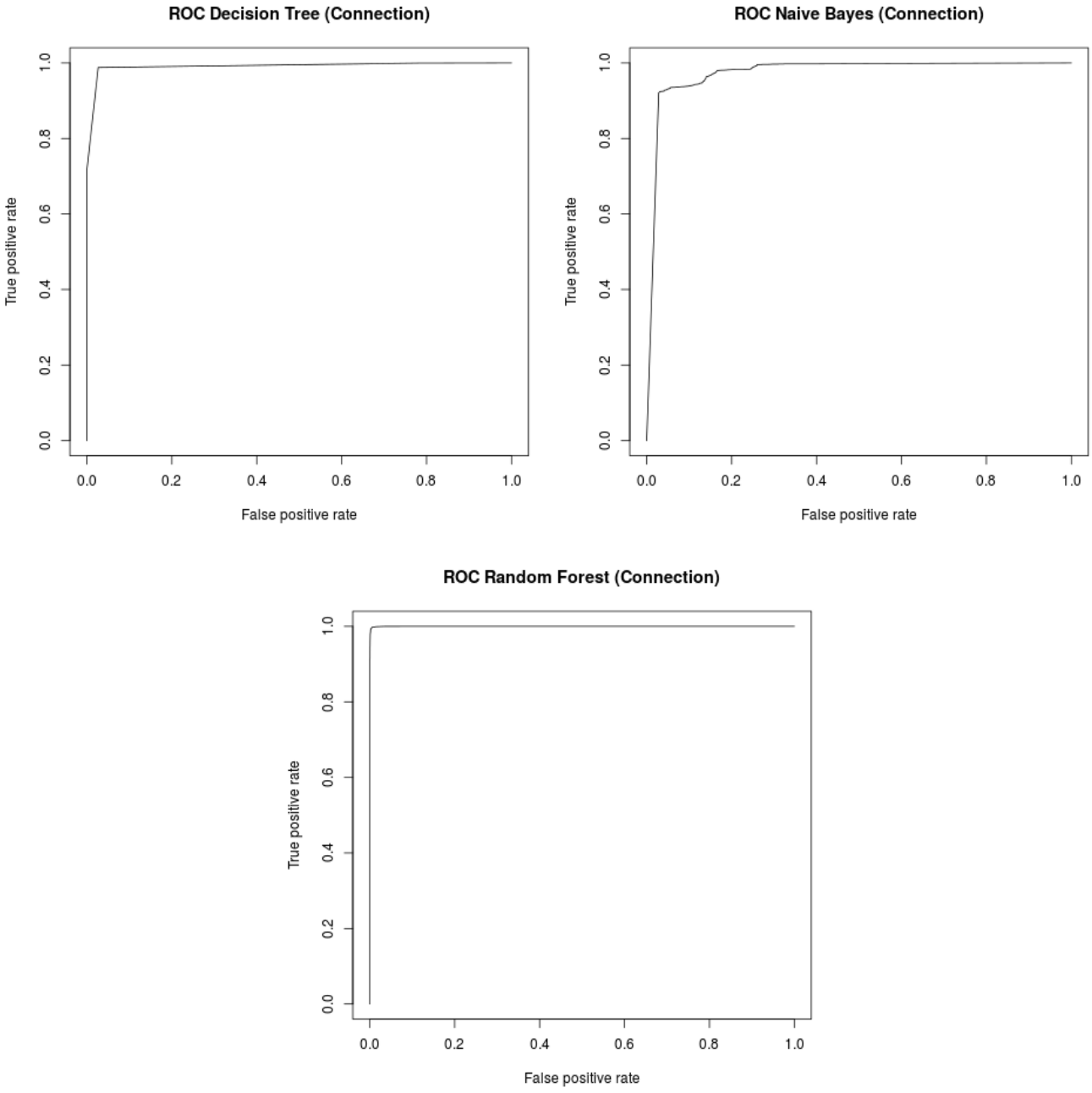

Figura 28. Curvas ROC modelos Connection

Chapter 5. Conclusiones y futuro

5.1. Cumplimiento de objetivos y planificación

Recapitulando todo lo realizado en el proyecto:

1. Se han propuesto diversas estrategias para la clasificación.
2. Se han localizado datasets de internet y se han preparado para su estudio.
3. Se ha desarrollado prácticamente de cero un software capaz de extraer hasta el más mínimo detalle de una conversación TLS y de almacenarla de forma estructurada en una base de datos documental.
4. Se han seleccionado y preparado los datos para diversos modelos de machine learning que contienen:
    a. Información de capacidades TLS ofrecidas por cliente y fingerprinting
    b. Información de todo el handshake: secuencias de handshakes, certificados, paquetes enviados distiguidos por protocolo… hasta del uso de fragmentación y la existencia de múltiples mensajes en un registro.
    c. Información del handshake+flujo: en este caso se almacenan también los bytes enviados, el tiempo de conexión, mensajes cifrados y tipo, en la conversación, etc.
5. Se han usado hasta cuatro algoritmos diferentes, probado y contrastado los modelos.
6. Se ha desarrollado un módulo para OpenCPU que exporta los modelos vía una API REST.
7. Se ha desarrollado prácticamente de cero un sistema IDS modular que hace uso de los modelos y dispone de funcionalidad adicional.

Si examinamos los objetivos que se fijaron en la primera PEC, se cumplen todos los objetivos mínimos, en el caso de objetivos deseables no hay multiclasificador porque finalmente no dispuse de datos agrupados para realizarlos y en lo que respecta a características de contexto (conexiones previas, DNS, etc), aunque el diseño del IDS lo permite, finalmente no se implementaron.

Siendo crítico con la planificación y los objetivos, fui excesivamente optimista y me pudo mi ambición por tratar de hacer algo diferente a lo que había visto en otros proyectos. Realmente no contaba con la experiencia de ningún proyecto parecido, desconocía completamente la minería de datos, no había programado casi nada en Go y R… en definitiva sobreestimé mi capacidad. Por si esto fuera poco tuve que buscar por mi cuenta todos los datos y no recibí feedback hasta una fase tardía del proyecto.

A pesar de esto, creo que he conseguido sacar adelante el proyecto y estoy orgulloso con el resultado.

5.2. Conclusiones

Si únicamente atendemos a los valores arrojados por las pruebas de validación de los modelos, éstos son simplemente espectaculares. Para el algoritmo y modelo con mejor clasificación (RandomForest - Connection) tenemos una precisión del **99,19%** y una tasa de falsos positivos del

0,14%. Y si observamos su curva ROC parece hecha con escuadra y cartabón.

Esto nos hace pensar que tenemos el mejor clasificador posible. A continuación ponemos la herramienta en un despliegue real y... detecta al primer Internet Explorer 9 que ve como malware. Tras volver a meter las botellas de cava en la nevera, la pregunta es ¿qué está pasando?

La metodología seguida ha sido clara: se han tomado capturas de distintas Botnets, capturas de tráfico normal, se han extraído los datos y mezclado pero no agitado en un dataset. Después se ha dividido de manera aleatoria en un `80%` para entrenamiento y un `20%` para validación. Esto es: ninguna de las muestras usadas para validación se ha usado para crear el modelo.

Podemos inmediatamente pensar que hay muchas muestras duplicadas y que nuestras muestras de validación "están contaminadas" ya que se han incorporado sus duplicados al modelo de entrenamiento. Sin embargo tras varias operaciones de unicidad se comprueba que esto no es así.

¿Entonces dónde está el problema? Si recapitulamos, el modelo fingerprint no es más que una reducción de nuestro universo de datos y, como vimos, sólo un 2% de los fingerprints coinciden para las dos clases: sólo en un 2% de los casos los atributos coinciden para malware y no malware. Dicho de otra manera, una instancia única de una serie de atributos de fingerprint es capaz de definir en el 98% de las muestras la clase a la que pertenece. Si tenemos en cuenta que las muestras usadas para el entrenamiento del modelo connection fueron `211496` mientras que el total de posibles combinaciones de fingerprint es de tan sólo `496` existe una enorme probabilidad de que en el conjunto de entrenamiento se encuentren todos los valores posibles del subconjunto de atributos que conforman el fingerprint y, por tanto los modelos sean capaces de clasificar con la enorme precisión que lo hacen ya que las "dudas" planteadas por las posibles coincidencias de fingerprints las resuelven gracias al aprendizaje obtenido a partir de las diferencias del resto de atributos.

Luego si probamos con fingerprints no existentes en el dataset, no funciona correctamente, lo que indica que al menos las relaciones que se han establecido internamente especialmente por los atributos de fingerprint no generalizan bien. Es muy revelador el árbol de decisión creado por el modelo fingerprint. El árbol que se muestra es el árbol del dataset final (tras la incorporación de todas las capturas). En él puede verse como una primera clasificación comprueba la existencia del algoritmo de firma con el código `2054`. Pues bien, si observamos las capturas del Windows Update, no ofrecía soporte a extensión por lo que en el anterior modelo no existía el aprendizaje de otras características clave que pudiese diferenciarlo de muestras etiquetadas como malware.

En resumen, nuestros modelos se encuentran sesgados por los atributos del fingerprint. Si observamos las combinaciones posibles de este subconjunto, éstas son reducidas y las generalizaciones las realiza en base a ausencia de extensiones, algoritmos de firma débiles, etc, que confirma la hipótesis de que gran parte del malware usa implementaciones "débiles" de TLS. Sin embargo esto tiene un grave efecto colateral y es que los modelos etiquetarán también al software antiguo (que no se haya incluído en el dataset) como malware. Por supuesto, esta generalización tampoco funcionará para detectar malware que imite o haga uso de librerías usadas por software actual y estándar.

5.3. Trabajo futuro

Como se ve, hay mucho trabajo por hacer. En primer lugar habría que eliminar los sesgos identificados y utilizar el resto de atributos que parecen tener menos fuerza para seguir extrayendo

conocimiento.

Aunque necesitaríamos muchas más muestras de fingerprints independientes, hemos visto que el fingerprint no es un buen clasificador ya que las generalizaciones que se puedan realizar del conjunto de datos tienen un recorrido más bien corto y en el momento que el malware "copie" uno existente pierde completamente su efectividad. Sin embargo el conjunto de estos atributos sí que puede ser un buen identificador o un buen preclasificador.

Posteriormente, suprimiendo estos atributos de fingerprint y empleando heurísticas (puerto de la conexión, información APLN, etc) se pueden crear diferentes modelos más específicos que traten de verificar el comportamiento. Para esto es fundamental un análisis y un modelado más específico que el realizado en este proyecto, ya que aquí sólo hemos agregado atributos resúmenes del flujo: bytes, duración, etc. En su lugar, empleando los datos obtenidos de la colección `records` habría que realizar otro tipo de modelos (posiblemente cadenas de Markov).

Por supuesto, habría que indagar en todas las propuestas para la clasificación que se sugirieron en la fase de análisis y que incluyen el uso del tráfico `DNS`, información de certificados y otro tipo de elementos.

Desde el punto de vista del software desarrollado, es necesario a corto plazo una refactorización, limpieza de código y solucionar algunos problemas de rendimiento y agregar varias optimizaciones que se dejaron para una fase posterior a la prueba de concepto.

Chapter 6. Glosario

API REST

La transferencia de estado representacional (en inglés representational state transfer) o REST es un estilo de arquitectura software para sistemas hipermedia distribuidos como la World Wide Web.

blacklist

Una lista negra o black list es una lista donde se registran las direcciones IPs que generan spam de forma voluntaria o involuntaria.

Botnet

Botnet es un término que hace referencia a un conjunto o red de robots informáticos o bots, que se ejecutan de manera autónoma y automática.

C&C

En el campo de seguridad informática, la infraestructura mando y control (Command and control en inglés, usualmente abreviado C&C o C2) consta de servidores y otros elementos que son usados para controlar los malware, tal con los, botnet.

cross validation o validación cruzada

La validación cruzada o cross-validation es una técnica utilizada para evaluar los resultados de un análisis estadístico y garantizar que son independientes de la partición entre datos de entrenamiento y prueba.

data minning

La minería de datos o exploración de datos (es la etapa de análisis de "Knowledge Discovery in Databases" o KDD) es un campo de la estadística y las ciencias de la computación referido al proceso que intenta descubrir patrones en grandes volúmenes de conjuntos de datos.

dataset

Un conjunto de datos (conocido también por el anglicismo: dataset, comúnmente utilizado en algunos países hispanohablantes) es una colección de datos habitualmente tabulada.

DNS

El sistema de nombres de dominio (DNS, por sus siglas en inglés, Domain Name System) es un sistema de nomenclatura jerárquico descentralizado para dispositivos conectados a redes IP como Internet o una red privada.

Entidades de Certificación o CA

Es aquella organización privada, que tiene como función evaluar la conformidad y certificar el cumplimiento de una norma de referencia, ya sea del producto, del servicio, o del sistema de gestión de una organización.

fingerprint

Conjunto de atributos que permiten identificar a un software.

hook

En programación, el término hooking abarca una gama de técnicas utilizadas para alterar o aumentar el comportamiento de un sistema operativo, de aplicaciones o de otros componentes de software interceptando llamadas de función o mensajes o eventos pasados entre componentes de software.

HTTP y HTTP/2

El Protocolo de transferencia de hipertexto (en inglés: Hypertext Transfer Protocol o HTTP) es el protocolo de comunicación que permite las transferencias de información en la World Wide Web. HTTP/2 es una evolución.

IP

Una dirección IP es un número que identifica, de manera lógica y jerárquica, a una Interfaz en red (elemento de comunicación/conexión) de un dispositivo (computadora, tableta, portátil, smartphone) que utilice el protocolo IP o (Internet Protocol), que corresponde al nivel de red del modelo TCP/IP.

LDAP

LDAP son las siglas de Lightweight Directory Access Protocol (en español Protocolo Ligero/Simplificado de Acceso a Directorios) que hacen referencia a un protocolo a nivel de aplicación que permite el acceso a un servicio de directorio ordenado y distribuido para buscar diversa información en un entorno de red.

malware

programa malicioso o programa maligno, también llamado badware, código maligno, software malicioso, software dañino o software malintencionado, es un tipo de software que tiene como objetivo infiltrarse o dañar una computadora o sistema de información.

Man-in-the-middle o MITM

es un ataque en el que se adquiere la capacidad de leer, insertar y modificar a voluntad.

NAT

La traducción de direcciones de red o NAT (del inglés Network Address Translation) es un mecanismo utilizado por routers IP para intercambiar paquetes entre dos redes que asignan mutuamente direcciones incompatibles.

NoSQL

Es una amplia clase de sistemas de gestión de bases de datos que difieren del modelo clásico de SGBDR (Sistema de Gestión de Bases de Datos Relacionales) en aspectos importantes, siendo el más destacado que no usan SQL como lenguaje principal de consultas.

PEC

Pruebas de Evaluación Contínua

Chapter 7. Referencias

- [1] https://blogs.cisco.com/security/detecting-encrypted-malware-traffic-without-decryption 2018-06-11
- [2] https://www.zscaler.com/ 2018-06-11
- [3] https://www.helpnetsecurity.com/2017/08/03/malicious-content-ssl-tls/ 2018-06-11
- [4] https://letsencrypt.org/ 2018-06-11
- [5] https://es.wikipedia.org/wiki/Cross_Industry_Standard_Process_for_Data_Mining 2018-06-11
- [6] Jordi Gironés, Jordi Casas, et al. Minería de datos, modelos y algoritmos, UOC, Barcelona, 2017
- [7] William Stallings, Lawrie Brown. Computer Security princpiples and practice, PEARSON, Harlow, 2015
- [8] https://tools.ietf.org/html/rfc5246 2018-06-11
- [9] https://tools.ietf.org/html/rfc4347 2018-06-11
- [10] https://cheapsslsecurity.com/blog/what-is-ssl-tls-handshake-understand-the-process-in-just-3-minutes/ 2018-06-11
- [11] https://tools.ietf.org/html/rfc4492 2018-06-11
- [12] http://citeseerx.ist.psu.edu/viewdoc/download?doi=10.1.1.110.1220&rep=rep1&type=pdf 2018-06-11
- [13] http://citeseerx.ist.psu.edu/viewdoc/download?doi=10.1.1.193.9242&rep=rep1&type=pdf 2018-06-11
- [14] https://www.researchgate.net/publication/289722788_Handling_Intrusion_Detection_System_using_Snort_Based_Statistical_Algorithm_and_Semi-supervised_Approach 2018-06-11
- [15] https://arxiv.org/abs/1710.04843 2018-06-11
- [16] https://arxiv.org/abs/1607.01639 2018-06-11
- [17] https://dl.acm.org/citation.cfm?id=2996768 2018-06-11
- [18] https://dl.acm.org/citation.cfm?id=3098163 2018-06-11
- [19] https://www.researchgate.net/profile/Sebastian_Garcia6/publication/271204142_Identifying_Modeling_and_Detecting_Botnet_Behaviors_in_the_Network/links/54c122a90cf2dd3cb95803b9/Identifying-Modeling-and-Detecting-Botnet-Behaviors-in-the-Network.pdf 2018-06-11
- [20] https://www.stratosphereips.org/ 2018-06-11
- [21] https://mcfp.weebly.com/the-ctu-13-dataset-a-labeled-dataset-with-botnet-normal-and-background-traffic.html 2018-06-11
- [22] https://mcfp.weebly.com/ 2018-06-11
- [23] https://mitmproxy.org/ 2018-06-11
- [24] http://www.tcpdump.org/ 2018-06-11

- [25] https://www.wireshark.org/ 2018-06-11
- [26] https://github.com/LeeBrotherston/tls-fingerprinting 2018-06-11
- [27] https://www.bro.org/ 2018-06-11
- [28] https://github.com/google/gopacket 2018-06-11
- [29] https://www.mongodb.com/ 2018-06-11
- [30] https://labix.org/mgo 2018-06-11
- [31] https://robomongo.org/ 2018-06-11
- [32] https://www.r-project.org/ 2018-06-11
- [33] https://www.rstudio.com/ 2018-06-11
- [34] https://www.cs.waikato.ac.nz/ml/weka/ 2018-06-11
- [35] https://www.opencpu.org/ 2018-06-11
- [36] https://dspace.cvut.cz/bitstream/handle/10467/68528/F3-BP-2017-Strasak-Frantisek-strasak_thesis_2017.pdf 2018-06-11
- [37] https://docs.mongodb.com/manual/aggregation/ 2018-06-11
- [38] https://github.com/jeroen/mongolite 2018-06-11

Appendix A: Características de los modelos

El modelo `Connection` contiene todas las características extraídas en este proyecto.

Muestra de características de modelo Connection


```
{
    "_id" : ObjectId("5b16b3fac4a3ea038896a95b"),
    "duration" : NumberLong(61811),
    "sendPackets" : NumberLong(10),
    "sendBytes" : NumberLong(9641),
    "sendPPS" : 0.162406176328659,
    "sendBPS" : 156.575790405273,
    "rcvdPackets" : NumberLong(8),
    "rcvdBytes" : NumberLong(3219),
    "rcvdPPS" : 0.1299207508564,
    "rcvdBPS" : 52.2768592834473,
    "clientHello-extensionlen" : 127,
    "clientHello-ciphersuitelen" : 30,
    "serverHello-extensionlen" : 13,
    "sendPlainAcc-hskrecords" : NumberLong(2),
    "sendPlainAcc-hskbytes" : NumberLong(272),
    "sendPlainAcc-cctrecords" : NumberLong(1),
    "sendPlainAcc-cctbytes" : NumberLong(1),
    "sendPlainAcc-appdatarecords" : NumberLong(0),
    "sendPlainAcc-appdatabytes" : NumberLong(0),
    "sendPlainAcc-fragmentedrecords" : 0,
    "sendPlainAcc-maxmessagesinrecord" : 1,
    "sendCipherAcc-hskrecords" : NumberLong(1),
    "sendCipherAcc-hskbytes" : NumberLong(40),
    "sendCipherAcc-cctrecords" : NumberLong(0),
    "sendCipherAcc-cctbytes" : NumberLong(0),
    "sendCipherAcc-appdatarecords" : NumberLong(1),
    "sendCipherAcc-appdatabytes" : NumberLong(9303),
    "rcvdPlainAcc-hskrecords" : NumberLong(2),
    "rcvdPlainAcc-hskbytes" : NumberLong(272),
    "rcvdPlainAcc-cctrecords" : NumberLong(1),
    "rcvdPlainAcc-cctbytes" : NumberLong(1),
    "rcvdPlainAcc-appdatarecords" : NumberLong(0),
    "rcvdPlainAcc-appdatabytes" : NumberLong(0),
    "rcvdPlainAcc-fragmentedrecords" : 0,
    "rcvdPlainAcc-maxmessagesinrecord" : 1,
    "rcvdCipherAcc-hskrecords" : NumberLong(1),
    "rcvdCipherAcc-hskbytes" : NumberLong(40),
    "rcvdCipherAcc-cctrecords" : NumberLong(0),
    "rcvdCipherAcc-cctbytes" : NumberLong(0),
    "rcvdCipherAcc-appdatarecords" : NumberLong(1),
    "rcvdCipherAcc-appdatabytes" : NumberLong(151),
    "clientHello-c_0" : false,
    "clientHello-c_3" : false,
```

```
    "clientHello-c_4" : false,
    "clientHello-c_5" : false,
    "clientHello-c_6" : false,
    "clientHello-c_7" : false,
    "clientHello-c_8" : false,
    "clientHello-c_9" : false,
    "clientHello-c_10" : true,
    "clientHello-c_13" : false,
    "clientHello-c_16" : false,
    "clientHello-c_17" : false,
    "clientHello-c_18" : false,
    "clientHello-c_19" : false,
    "clientHello-c_20" : false,
    "clientHello-c_21" : false,
    "clientHello-c_22" : false,
    "clientHello-c_47" : true,
    "clientHello-c_48" : false,
    "clientHello-c_49" : false,
    "clientHello-c_50" : false,
    "clientHello-c_51" : true,
    "clientHello-c_53" : true,
    "clientHello-c_54" : false,
    "clientHello-c_55" : false,
    "clientHello-c_56" : false,
    "clientHello-c_57" : true,
    "clientHello-c_60" : false,
    "clientHello-c_61" : false,
    "clientHello-c_62" : false,
    "clientHello-c_63" : false,
    "clientHello-c_64" : false,
    "clientHello-c_65" : false,
    "clientHello-c_68" : false,
    "clientHello-c_69" : false,
    "clientHello-c_98" : false,
    "clientHello-c_99" : false,
    "clientHello-c_100" : false,
    "clientHello-c_102" : false,
    "clientHello-c_103" : false,
    "clientHello-c_104" : false,
    "clientHello-c_105" : false,
    "clientHello-c_106" : false,
    "clientHello-c_107" : false,
    "clientHello-c_132" : false,
    "clientHello-c_135" : false,
    "clientHello-c_136" : false,
    "clientHello-c_150" : false,
    "clientHello-c_153" : false,
    "clientHello-c_154" : false,
    "clientHello-c_156" : false,
    "clientHello-c_157" : false,
    "clientHello-c_158" : false,
```

```
    "clientHello-c_159" : false,
    "clientHello-c_160" : false,
    "clientHello-c_161" : false,
    "clientHello-c_162" : false,
    "clientHello-c_163" : false,
    "clientHello-c_164" : false,
    "clientHello-c_165" : false,
    "clientHello-c_186" : false,
    "clientHello-c_189" : false,
    "clientHello-c_190" : false,
    "clientHello-c_192" : false,
    "clientHello-c_195" : false,
    "clientHello-c_196" : false,
    "clientHello-c_255" : false,
    "clientHello-c_4865" : false,
    "clientHello-c_4866" : false,
    "clientHello-c_4867" : false,
    "clientHello-c_22016" : false,
    "clientHello-c_23130" : false,
    "clientHello-c_49154" : false,
    "clientHello-c_49155" : false,
    "clientHello-c_49156" : false,
    "clientHello-c_49157" : false,
    "clientHello-c_49159" : false,
    "clientHello-c_49160" : false,
    "clientHello-c_49161" : true,
    "clientHello-c_49162" : true,
    "clientHello-c_49164" : false,
    "clientHello-c_49165" : false,
    "clientHello-c_49166" : false,
    "clientHello-c_49167" : false,
    "clientHello-c_49169" : false,
    "clientHello-c_49170" : false,
    "clientHello-c_49171" : true,
    "clientHello-c_49172" : true,
    "clientHello-c_49174" : false,
    "clientHello-c_49175" : false,
    "clientHello-c_49176" : false,
    "clientHello-c_49177" : false,
    "clientHello-c_49179" : false,
    "clientHello-c_49180" : false,
    "clientHello-c_49182" : false,
    "clientHello-c_49183" : false,
    "clientHello-c_49185" : false,
    "clientHello-c_49186" : false,
    "clientHello-c_49187" : false,
    "clientHello-c_49188" : false,
    "clientHello-c_49189" : false,
    "clientHello-c_49190" : false,
    "clientHello-c_49191" : false,
    "clientHello-c_49192" : false,
```

```
    "clientHello-c_49193" : false,
    "clientHello-c_49194" : false,
    "clientHello-c_49195" : true,
    "clientHello-c_49196" : true,
    "clientHello-c_49197" : false,
    "clientHello-c_49198" : false,
    "clientHello-c_49199" : true,
    "clientHello-c_49200" : true,
    "clientHello-c_49201" : false,
    "clientHello-c_49202" : false,
    "clientHello-c_49266" : false,
    "clientHello-c_49267" : false,
    "clientHello-c_49270" : false,
    "clientHello-c_49271" : false,
    "clientHello-c_49274" : false,
    "clientHello-c_49275" : false,
    "clientHello-c_49276" : false,
    "clientHello-c_49277" : false,
    "clientHello-c_49280" : false,
    "clientHello-c_49281" : false,
    "clientHello-c_49286" : false,
    "clientHello-c_49287" : false,
    "clientHello-c_49290" : false,
    "clientHello-c_49291" : false,
    "clientHello-c_52243" : false,
    "clientHello-c_52244" : false,
    "clientHello-c_52245" : false,
    "clientHello-c_52392" : true,
    "clientHello-c_52393" : true,
    "clientHello-c_64250" : false,
    "clientHello-c_65279" : false,
    "serverHello-c_0" : false,
    "serverHello-c_3" : false,
    "serverHello-c_4" : false,
    "serverHello-c_5" : false,
    "serverHello-c_6" : false,
    "serverHello-c_7" : false,
    "serverHello-c_8" : false,
    "serverHello-c_9" : false,
    "serverHello-c_10" : false,
    "serverHello-c_13" : false,
    "serverHello-c_16" : false,
    "serverHello-c_17" : false,
    "serverHello-c_18" : false,
    "serverHello-c_19" : false,
    "serverHello-c_20" : false,
    "serverHello-c_21" : false,
    "serverHello-c_22" : false,
    "serverHello-c_47" : false,
    "serverHello-c_48" : false,
    "serverHello-c_49" : false,
```

```
    "serverHello-c_50" : false,
    "serverHello-c_51" : false,
    "serverHello-c_53" : false,
    "serverHello-c_54" : false,
    "serverHello-c_55" : false,
    "serverHello-c_56" : false,
    "serverHello-c_57" : false,
    "serverHello-c_60" : false,
    "serverHello-c_61" : false,
    "serverHello-c_62" : false,
    "serverHello-c_63" : false,
    "serverHello-c_64" : false,
    "serverHello-c_65" : false,
    "serverHello-c_68" : false,
    "serverHello-c_69" : false,
    "serverHello-c_98" : false,
    "serverHello-c_99" : false,
    "serverHello-c_100" : false,
    "serverHello-c_102" : false,
    "serverHello-c_103" : false,
    "serverHello-c_104" : false,
    "serverHello-c_105" : false,
    "serverHello-c_106" : false,
    "serverHello-c_107" : false,
    "serverHello-c_132" : false,
    "serverHello-c_135" : false,
    "serverHello-c_136" : false,
    "serverHello-c_150" : false,
    "serverHello-c_153" : false,
    "serverHello-c_154" : false,
    "serverHello-c_156" : false,
    "serverHello-c_157" : false,
    "serverHello-c_158" : false,
    "serverHello-c_159" : false,
    "serverHello-c_160" : false,
    "serverHello-c_161" : false,
    "serverHello-c_162" : false,
    "serverHello-c_163" : false,
    "serverHello-c_164" : false,
    "serverHello-c_165" : false,
    "serverHello-c_186" : false,
    "serverHello-c_189" : false,
    "serverHello-c_190" : false,
    "serverHello-c_192" : false,
    "serverHello-c_195" : false,
    "serverHello-c_196" : false,
    "serverHello-c_255" : false,
    "serverHello-c_4865" : false,
    "serverHello-c_4866" : false,
    "serverHello-c_4867" : false,
    "serverHello-c_22016" : false,
```

```
    "serverHello-c_23130" : false,
    "serverHello-c_49154" : false,
    "serverHello-c_49155" : false,
    "serverHello-c_49156" : false,
    "serverHello-c_49157" : false,
    "serverHello-c_49159" : false,
    "serverHello-c_49160" : false,
    "serverHello-c_49161" : false,
    "serverHello-c_49162" : false,
    "serverHello-c_49164" : false,
    "serverHello-c_49165" : false,
    "serverHello-c_49166" : false,
    "serverHello-c_49167" : false,
    "serverHello-c_49169" : false,
    "serverHello-c_49170" : false,
    "serverHello-c_49171" : false,
    "serverHello-c_49172" : false,
    "serverHello-c_49174" : false,
    "serverHello-c_49175" : false,
    "serverHello-c_49176" : false,
    "serverHello-c_49177" : false,
    "serverHello-c_49179" : false,
    "serverHello-c_49180" : false,
    "serverHello-c_49182" : false,
    "serverHello-c_49183" : false,
    "serverHello-c_49185" : false,
    "serverHello-c_49186" : false,
    "serverHello-c_49187" : false,
    "serverHello-c_49188" : false,
    "serverHello-c_49189" : false,
    "serverHello-c_49190" : false,
    "serverHello-c_49191" : false,
    "serverHello-c_49192" : false,
    "serverHello-c_49193" : false,
    "serverHello-c_49194" : false,
    "serverHello-c_49195" : false,
    "serverHello-c_49196" : false,
    "serverHello-c_49197" : false,
    "serverHello-c_49198" : false,
    "serverHello-c_49199" : true,
    "serverHello-c_49200" : false,
    "serverHello-c_49201" : false,
    "serverHello-c_49202" : false,
    "serverHello-c_49266" : false,
    "serverHello-c_49267" : false,
    "serverHello-c_49270" : false,
    "serverHello-c_49271" : false,
    "serverHello-c_49274" : false,
    "serverHello-c_49275" : false,
    "serverHello-c_49276" : false,
    "serverHello-c_49277" : false,
```

```
    "serverHello-c_49280" : false,
    "serverHello-c_49281" : false,
    "serverHello-c_49286" : false,
    "serverHello-c_49287" : false,
    "serverHello-c_49290" : false,
    "serverHello-c_49291" : false,
    "serverHello-c_52243" : false,
    "serverHello-c_52244" : false,
    "serverHello-c_52245" : false,
    "serverHello-c_52392" : false,
    "serverHello-c_52393" : false,
    "serverHello-c_64250" : false,
    "serverHello-c_65279" : false,
    "clientHello-e_0" : true,
    "clientHello-e_5" : true,
    "clientHello-e_10" : true,
    "clientHello-e_11" : true,
    "clientHello-e_13" : true,
    "clientHello-e_15" : false,
    "clientHello-e_16" : true,
    "clientHello-e_18" : false,
    "clientHello-e_21" : false,
    "clientHello-e_23" : true,
    "clientHello-e_35" : true,
    "clientHello-e_41" : false,
    "clientHello-e_42" : false,
    "clientHello-e_43" : false,
    "clientHello-e_45" : false,
    "clientHello-e_51" : false,
    "clientHello-e_13172" : false,
    "clientHello-e_14906" : false,
    "clientHello-e_30031" : false,
    "clientHello-e_30032" : false,
    "clientHello-e_35466" : false,
    "clientHello-e_35655" : false,
    "clientHello-e_56026" : false,
    "clientHello-e_60138" : false,
    "clientHello-e_65281" : true,
    "clientHello-e_65283" : false,
    "clientHello-e_0_len" : 35,
    "clientHello-e_5_len" : 5,
    "clientHello-e_10_len" : 10,
    "clientHello-e_11_len" : 2,
    "clientHello-e_13_len" : 24,
    "clientHello-e_15_len" : 0,
    "clientHello-e_16_len" : 14,
    "clientHello-e_18_len" : 0,
    "clientHello-e_21_len" : 0,
    "clientHello-e_23_len" : 0,
    "clientHello-e_35_len" : 0,
    "clientHello-e_41_len" : 0,
```

```
    "clientHello-e_42_len" : 0,
    "clientHello-e_43_len" : 0,
    "clientHello-e_45_len" : 0,
    "clientHello-e_51_len" : 0,
    "clientHello-e_13172_len" : 0,
    "clientHello-e_14906_len" : 0,
    "clientHello-e_30031_len" : 0,
    "clientHello-e_30032_len" : 0,
    "clientHello-e_35466_len" : 0,
    "clientHello-e_35655_len" : 0,
    "clientHello-e_56026_len" : 0,
    "clientHello-e_60138_len" : 0,
    "clientHello-e_65281_len" : 1,
    "clientHello-e_65283_len" : 0,
    "serverHello-e_0" : false,
    "serverHello-e_5" : false,
    "serverHello-e_10" : false,
    "serverHello-e_11" : true,
    "serverHello-e_13" : false,
    "serverHello-e_15" : false,
    "serverHello-e_16" : false,
    "serverHello-e_18" : false,
    "serverHello-e_21" : false,
    "serverHello-e_23" : false,
    "serverHello-e_35" : false,
    "serverHello-e_41" : false,
    "serverHello-e_42" : false,
    "serverHello-e_43" : false,
    "serverHello-e_45" : false,
    "serverHello-e_51" : false,
    "serverHello-e_13172" : false,
    "serverHello-e_14906" : false,
    "serverHello-e_30031" : false,
    "serverHello-e_30032" : false,
    "serverHello-e_35466" : false,
    "serverHello-e_35655" : false,
    "serverHello-e_56026" : false,
    "serverHello-e_60138" : false,
    "serverHello-e_65281" : true,
    "serverHello-e_65283" : false,
    "serverHello-e_0_len" : 0,
    "serverHello-e_5_len" : 0,
    "serverHello-e_10_len" : 0,
    "serverHello-e_11_len" : 4,
    "serverHello-e_13_len" : 0,
    "serverHello-e_15_len" : 0,
    "serverHello-e_16_len" : 0,
    "serverHello-e_18_len" : 0,
    "serverHello-e_21_len" : 0,
    "serverHello-e_23_len" : 0,
    "serverHello-e_35_len" : 0,
```

```
    "serverHello-e_41_len" : 0,
    "serverHello-e_42_len" : 0,
    "serverHello-e_43_len" : 0,
    "serverHello-e_45_len" : 0,
    "serverHello-e_51_len" : 0,
    "serverHello-e_13172_len" : 0,
    "serverHello-e_14906_len" : 0,
    "serverHello-e_30031_len" : 0,
    "serverHello-e_30032_len" : 0,
    "serverHello-e_35466_len" : 0,
    "serverHello-e_35655_len" : 0,
    "serverHello-e_56026_len" : 0,
    "serverHello-e_60138_len" : 0,
    "serverHello-e_65281_len" : 1,
    "serverHello-e_65283_len" : 0,
    "clientHello-oscp" : true,
    "clientHello-sa_257" : false,
    "clientHello-sa_513" : true,
    "clientHello-sa_514" : false,
    "clientHello-sa_515" : true,
    "clientHello-sa_769" : false,
    "clientHello-sa_770" : false,
    "clientHello-sa_771" : false,
    "clientHello-sa_1025" : true,
    "clientHello-sa_1026" : false,
    "clientHello-sa_1027" : true,
    "clientHello-sa_1281" : true,
    "clientHello-sa_1282" : false,
    "clientHello-sa_1283" : true,
    "clientHello-sa_1537" : true,
    "clientHello-sa_1538" : false,
    "clientHello-sa_1539" : true,
    "clientHello-sa_2052" : true,
    "clientHello-sa_2053" : true,
    "clientHello-sa_2054" : true,
    "clientHello-sg_1" : false,
    "clientHello-sg_2" : false,
    "clientHello-sg_3" : false,
    "clientHello-sg_4" : false,
    "clientHello-sg_5" : false,
    "clientHello-sg_6" : false,
    "clientHello-sg_7" : false,
    "clientHello-sg_8" : false,
    "clientHello-sg_9" : false,
    "clientHello-sg_10" : false,
    "clientHello-sg_11" : false,
    "clientHello-sg_12" : false,
    "clientHello-sg_13" : false,
    "clientHello-sg_14" : false,
    "clientHello-sg_15" : false,
    "clientHello-sg_16" : false,
```

```
    "clientHello-sg_17" : false,
    "clientHello-sg_18" : false,
    "clientHello-sg_19" : false,
    "clientHello-sg_20" : false,
    "clientHello-sg_21" : false,
    "clientHello-sg_22" : false,
    "clientHello-sg_23" : true,
    "clientHello-sg_24" : true,
    "clientHello-sg_25" : true,
    "clientHello-sg_26" : false,
    "clientHello-sg_27" : false,
    "clientHello-sg_28" : false,
    "clientHello-sg_29" : true,
    "clientHello-sg_256" : false,
    "clientHello-sg_257" : false,
    "clientHello-sg_51914" : false,
    "clientHello-sg_60138" : false,
    "clientHello-alpn_h2" : true,
    "clientHello-alpn_h214" : false,
    "clientHello-alpn_h215" : false,
    "clientHello-alpn_h216" : false,
    "clientHello-alpn_http11" : true,
    "clientHello-alpn_spdy2" : false,
    "clientHello-alpn_spdy3" : false,
    "clientHello-alpn_spdy31" : false,
    "serverHello-alpn_h2" : false,
    "serverHello-alpn_h214" : false,
    "serverHello-alpn_h215" : false,
    "serverHello-alpn_h216" : false,
    "serverHello-alpn_http11" : false,
    "serverHello-alpn_spdy2" : false,
    "serverHello-alpn_spdy3" : false,
    "serverHello-alpn_spdy31" : false,
    "sendHskSeq_0" : false,
    "sendHskSeq_1" : false,
    "sendHskSeq_16" : false,
    "sendHskSeq_33" : true,
    "sendHskSeq_71" : false,
    "rcvdHskSeq_0" : false,
    "rcvdHskSeq_2" : false,
    "rcvdHskSeq_4" : false,
    "rcvdHskSeq_10" : false,
    "rcvdHskSeq_24" : false,
    "rcvdHskSeq_60" : false,
    "rcvdHskSeq_66" : false,
    "rcvdHskSeq_77" : false,
    "rcvdHskSeq_82" : false,
    "rcvdHskSeq_116" : true,
    "rcvdHskSeq_119" : false,
    "rcvdHskSeq_136" : false,
    "rcvdHskSeq_146" : false,
```

```
    "rcvdHskSeq_166" : false,
    "rcvdHskSeq_208" : false,
    "rcvdHskSeq_232" : false,
    "clientHello-version_768" : false,
    "clientHello-version_769" : false,
    "clientHello-version_770" : false,
    "clientHello-version_771" : true,
    "clientHello-sessionidlen_0" : true,
    "clientHello-sessionidlen_16" : false,
    "clientHello-sessionidlen_32" : false,
    "serverHello-version_768" : false,
    "serverHello-version_769" : false,
    "serverHello-version_770" : false,
    "serverHello-version_771" : true,
    "serverHello-sessionidlen_0" : false,
    "serverHello-sessionidlen_16" : false,
    "serverHello-sessionidlen_32" : true
}
```