TRABAJO FIN DE GRADO

INGENIERÍA DE TECNOLOGÍAS DE TELECOMUNICACIÓN

Módulo de análisis de ofuscación de código JavaScript para sistema de detección de ataques Drive-By-Download

Autor

Lucas Cruz Cruz

Director

Gabriel Maciá Fernández

Escuela Técnica Superior de Ingenierías Informática y de Telecomunicación

—

Granada, Junio de 2016

Módulo de análisis de ofuscación de código JavaScript para sistema de detección de ataques Drive-By-Download

Autor
Lucas Cruz Cruz

Director
Gabriel Maciá Fernández

Módulo de análisis de ofuscación de código JavaScript para sistema de detección de ataques Drive-By-Download

Lucas Cruz Cruz

Palabras clave: seguridad, ofuscación, javascript, navegación web, htmlunit.

Resumen

El uso de Internet, y por ende, de todos los servicios que hacen uso del mismo, ha crecido de manera exponencial en los últimos años. La gran mayoría de los hogares en nuestro país dispone hoy día de más de una conexión a Internet (véase ordenador de sobremesa, portátil, móviles, tablets, etc.) y muchas de las gestiones que hace menos de una década se realizaban de manera presencial y/o en papel se han pasado al bando digital.

En un principio todo parecen ser ventajas: no necesitas ir al banco para ejecutar una transferencia, puedes pedir compra desde el sofá o realizar aquel curso que bien podría valerte un ascenso pero que se impartía a cientos de kilómetros de casa. Pero la realidad es bien distinta, todas estas acciones online suponen la puesta en circulación de millones y millones de datos personales e información sensible que, de ser sustraída, puede acarrear un grave problema para su dueño así como una gran suma para la cuenta bancaria de los delincuentes cibernéticos.

Estos delincuentes tienen muchas y diversas formas de alcanzar su objetivo, casi todas basadas en el engaño (phising es una de las más conocidas) o la infección de sistemas con malware de distinta naturaleza. Este proyecto se centrará en una de las técnicas más peligrosas, por su difícil detección si no se cuenta con las medidas de protección oportunas, y utilizadas: los ataques Drive-By-Download.

Los ataques de Drive-by-Download suponen en la actualidad el principal vector de infección con malware de sistemas en Internet. Estos ataques se implementan inyectando de forma maliciosa código JavaScript en páginas web legítimas o directamente mediante la creación de webs que actúen a modo de cebo, de tal manera que al ser descargadas por los usuarios dicho código malicioso evalúe las vulnerabilidades del navegador e infecte el sistema. O lo que es lo mismo, en caso de tener éxito, vía libre a toda la información del mismo.

En este proyecto se pretende desarrollar un módulo para la detección de código JavaScript ofuscado y su integración en un sistema para la detección de ataques Drive-by-Download. Para ello, el sistema deberá monitorizar la descarga de páginas web para detectar patrones de ofuscación en el código JavaScript que apunten a la existencia de código malicioso. En otras palabras, será capaz de emular el comportamiento de un navegador normal al al descargar las páginas web pero deberá decidir si se podría estar sufriendo un ataque Drive-by-Download basándonos en esta característica

Analysis module of obfuscated JavaScript code for Drive-by-Download attacks detection system

Lucas Cruz Cruz

Keywords: security, obfuscation, javascript, web browsing, htmlunit.

Abstract

Internet use, and therefore, all the services that make use of it, has grown exponentially in the last years. Nowadays, the most of the homes in our country has more than one Internet connection (computer, laptop, phones, tablets, etc.) and many of the managements that less than a decade ago were made in person and/or paper has changed to the digital side.

At first, everything looks like advantages: you don't need to go to the bank to perform a transfer, you can order purchase from the couch or take that course that could avail yourself a promotion but was taught hundreds of kilometers from home. But the reality is quite different, all these online activities involve the circulation of thousands of personal data and sensitive information that, if stolen, can lead to a serious problem for its owner as well as a large sum for the bank account of the cybercriminals.

These criminals have many different ways to achieve their goal, almost all based on deception (phishing is one of the best known) or infection with malware systems of different nature. This project will focus on one of the most dangerous techniques, for its difficult detection if you do not have the appropriate protection measures, and used: the Drive-By-Download attacks.

Attacks Drive-by-Download are now the main infection vector with malware of systems on the Internet. These attacks are implemented by injecting maliciously JavaScript code into legitimate web pages or by creating malicious websites, so that to be downloaded by users that malware evaluate browser vulnerabilities and infect the system. Or what is the same, if successful, free to all information the same way.

This project aims to develop a module to detect obfuscated JavaScript code and its integration into a system for detecting attacks Drive-by-Download. To do this, the system should monitor downloading web pages to detect patterns in JavaScript code obfuscation point to the existence of malicious code. In other words, it will be able to emulate the behavior of a normal to downloading web pages but it has to decide whether you might be suffering from a Drive-by-Download attack based on this feature.

Yo, **Lucas Cruz Cruz**, alumno de la titulación Grado en Ingeniería de las Tecnologías de Telecomunicación de la **Escuela Técnica Superior de Ingenierías Informática y de Telecomunicación de la Universidad de Granada**, con DNI 75119440Z, autorizo la ubicación de la siguiente copia de mi Trabajo Fin de Grado en la biblioteca del centro para que pueda ser consultada por las personas que lo deseen.

Fdo: Lucas Cruz Cruz

Granada a 20 de Junio de 2016.

D. **Gabriel Maciá Fernández**, Profesor del Departamento Teoría de la Señal, telemática y Comunicaciones de la Universidad de Granada.

Informa:

Que el presente trabajo, titulado: **Módulo de análisis de ofuscación de código JavaScript para sistema de detección de ataques Drive-By-Download** ha sido realizado bajo su supervisión por **Lucas Cruz Cruz**, y autorizo la defensa de dicho trabajo ante el tribunal que corresponda.

Y para que conste, expiden y firman el presente informe en Granada a 20 de Junio de 2016.

Los directores:

Gabriel Maciá Fernández

Agradecimientos

Antes de comenzar la memoria, me gustaría agradecer su apoyo a todas aquellas personas que de algún modo han colaborado conmigo en el desarrollo de este Trabajo Fin de Grado. En especial, a Gabriel Maciá Fernández, por aceptar la propuesta de ser mi tutor y llevar a cabo este Trabajo Fin de Grado, por todos los conocimientos prestados así como su disposición, paciencia y ayuda ante cualquier duda.

También, agradecer a mis padres el apoyo económico recibido durante estos años de carrera. Por último, gracias a todos mis compañeros y amigos que de una forma u otra se han implicado en el desarrollo del mismo brindándome su apoyo, sobre todo a mi novia, Ana Belén, sin la cual esto no hubiera sido posible.

ÍNDICE

1. INTRODUCCIÓN

1.1. ATAQUES DRIVE-BY-DOWNLOAD

Internet llegó y lo cambió todo. Aunque hoy día ya estamos totalmente habituados a su uso, hasta el punto que parece casi imposible vivir sin todo lo que nos ofrece, no hace tanto las cosas funcionaban de manera muy diferente.

En la última década hemos vivido una abrupta transición hacia lo digital, lo que ha originado que muchos sectores hayan cambiado totalmente su forma de operar. Pero en esta lista, no solo se incluyen el comercio, la banca o las comunicaciones, también están presentes aquellos dedicados a diversas actividades delictivas. De igual manera que la mayor parte de la información ya no se guarda en papeles ni es necesaria una cartera para llevar dinero, un ladrón no precisa de un cuchillo y un pasamontañas para robarte ni una banda organizada un arsenal armamentístico para realizar atracos a gran escala.

Lo que necesitan es información, y un método para acceder a ella. Actualmente no son pocas las técnicas avanzadas que existen para la propagación masiva de códigos maliciosos que no solo se limitan a diseminar malware a través de spam y clientes de mensajería instantánea, sino que además lo hacen a través del acceso a determinado sitio o página web.

Un ejemplo de esto lo constituye la metodología de ataque denominada Drive-by-Download, una técnica aplicada por los ciberdelincuentes para propagar malware mediante la creación de webs maliciosas u opcionalmente, aprovechándose de las vulnerabilidades existentes en diferentes sitios web, las cuales, una vez explotadas, permiten la inyección de código malicioso entre el código legítimo del sitio. El objetivo no es otro infectar los equipos de aquellos usuarios que visiten dicha web comprometida, aprovechando también las vulnerabilidades del navegador e iniciando la descarga e instalación automática del malware sin el conocimiento o permiso del usuario. Esta técnica aplicada de manera repetida en múltiples sitios web permite la infección de un gran número de sistemas sin que la mayoría de los afectados sean siquiera conscientes.

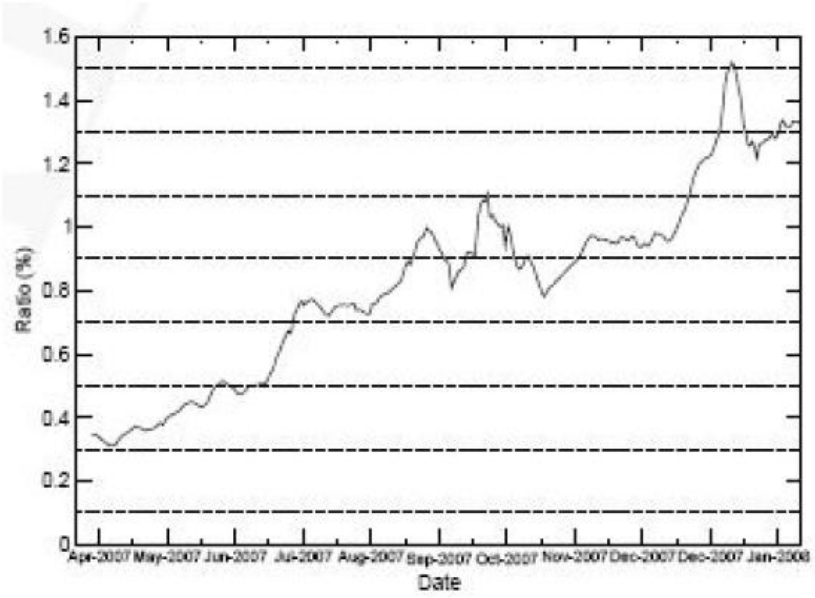

Figura 1: Resultados de búsquedas con URL comprometidas en 2007[1]

Aunque como se mencionaba anteriormente, existen múltiples técnicas para alcanzar un mismo objetivo, se ha registrado un incremento significativo de ataques Drive-by-Download en la última década (la 'Figura 1: Resultados de búsquedas con URL comprometidas en 2007' muestra uno de los crecimientos más destacados, registrado en el año 2007). En este tiempo se han contabilizado numerosas víctimas, algunas de ellas ilustres como la web del gobierno del Reino Unido y las de las compañías de seguridad online F-Secure y Kaspersky.

Su capacidad de dispersión (no hay límite a la hora de infectar sitios web aparentemente seguros), sus posibilidades para infundir desconfianza (ningún sitio está totalmente a salvo de sufrir un ataque y por tanto, no es posible una navegación 100% segura), así como su notable índice de éxito, convierten a los ataques Drive-by-Download en uno de los mayores retos actuales en términos de seguridad web y por tanto, en una técnica muy interesante de ser estudiada, aún con todo lo que ya sabemos sobre ella.

1.1.1. FASES DE UN ATAQUE TÍPICO

Lo primero que se ha de tener claro son las fases de ejecución de un ataque Drive-by-Download. La 'Figura 2: Esquema de un ataque Drive-by-Download',

[1] Figura obtenida de (Niki, 2008-2009)

muestra gráficamente cada uno de los puntos que se van a desarrollar de manera individual más abajo.

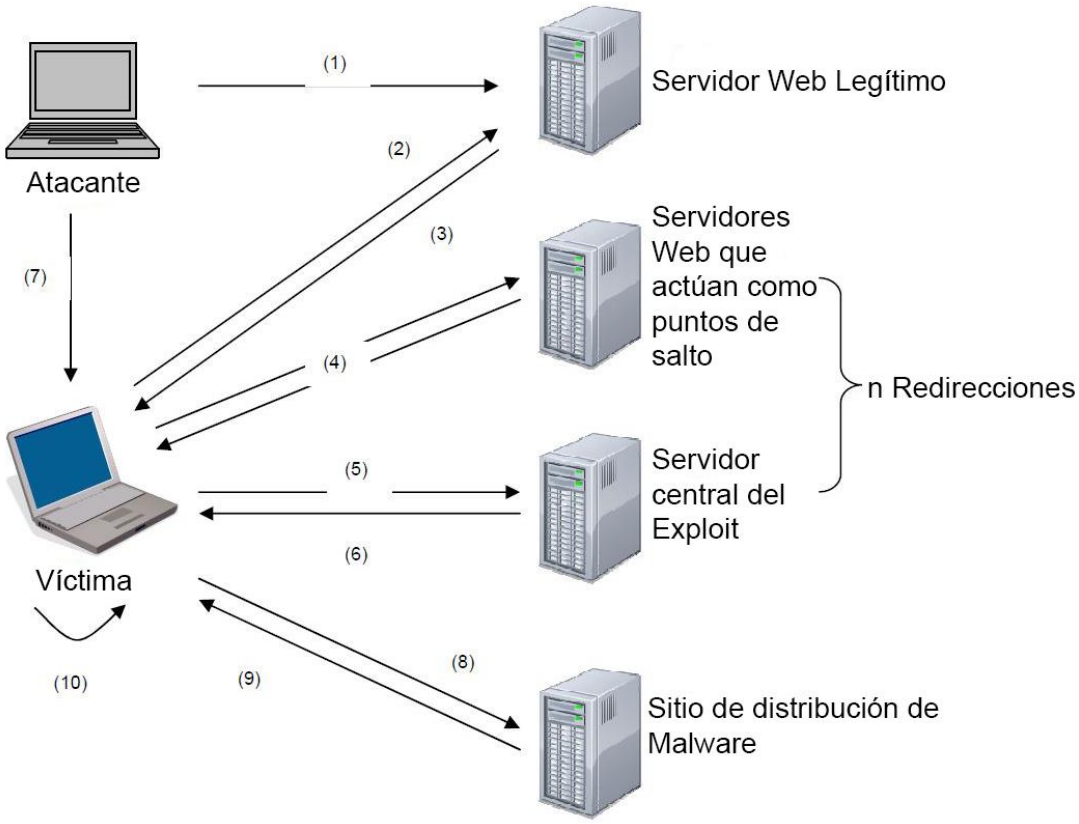

Figura 2: Esquema de un ataque Drive-by-Download

1) El atacante compromete un servidor web legítimo aprovechándose de alguna vulnerabilidad e inserta una secuencia de comandos en una aplicación web. En este instante el sitio web ya está comprometido, y de no ser detectado, el atacante estará listo para la llegada de sus víctimas. Este primer paso no siempre es necesario, ya que el atacante puede hacer uso de un sitio web de su propiedad.

2) La víctima visita el sitio web que ha sido comprometido o creado por el atacante. En un principio, nada extraño, es posible que el sitio web sea una de tus visitas habituales, no notarás nada, pero esta vez puede ocasionarte un problema grave.

3) El servidor web envía, junto con la página solicitada, la secuencia de comandos del atacante. Este script puede ser ejecutado o bien introduce una secuencia de comandos (puede ser en forma de otro script que lo importa desde un servidor central). Aquí tenemos dos opciones, esta

importación puede incluir directamente los recursos del servidor remoto o bien una orden para seguir una serie de redirecciones.

4) En el caso de que sea la segunda opción (más habitual), se inicia la secuencia de redirecciones desde un servidor web a otro que en realidad desempeñan el papel de puntos de salto. Lo importante para el atacante en este punto es difuminar al máximo posible las huellas dejadas para que realizar un seguimiento sea casi imposible.
5) Después de ejecutar un número determinado (n) de redirecciones de la víctima llega al servidor central del exploit. Sea en este o en otro de los servidores, es común que se compruebe si el sistema ha sido atacado antes mediante el uso de una cookie, esto sirve para evitar ataques innecesarios, reduciendo el riesgo de ser localizado.
6) El servidor envía el exploit en un script ofuscado. En función de la información obtenida en los pasos anteriores (sistema operativo, navegador, versión del navegador, etc.) se selecciona un exploit u otro.
7) Una vez es ejecutado el código, y si todo sale bien, el atacante tomará el control del sistema de la víctima, siempre después de explotar alguna vulnerabilidad.
8) El exploit instruye al navegador para visitar el sitio de distribución de malware, lugar donde se encuentra realmente el vector de infección, y no, como pudiera parecer a simple vista, en el sitio original que ha sido comprometido. Llegados aquí, comienza realmente el ataque Drive-by Download.
9) Los ejecutables de malware se descargan.
10) El equipo de la víctima instala automáticamente y ejecuta el código malicioso.

1.1.2. CARACTERÍSTICAS

A pesar de todo lo mencionado anteriormente, los ataques Drive-by-Download están lejos de ser perfectos. Como es de esperar, un sitio web que

haya sido comprometido tendrá un comportamiento distinto al que tendría si funcionara de forma normal. Ya se han explicado cuáles sus fases, por lo que bastaría con analizar los eventos que se generan durante un tiempo y comprobar si existe alguna anomalía para determinar si un sitio web está o no limpio.

Es en este punto cuando nos preguntamos ¿qué es una anomalía? Para poder responder es necesario extraer una serie de "características normales" mediante ensayo y aprendizaje que darán como resultado un modelo de comportamiento. Será entonces cuando se puedan observar diferentes desviaciones de este comportamiento normal para decidir si se está actuando tal y como cabría esperar o por el contrario, hay algún indicador de que el sitio en cuestión podría estar comprometido.

Siguiendo este *modus operandi* algunos estudios[2] han establecido en cinco las características de un ataque Drive-by-Download típico.

1.1.2.1. Redirección y cloaking

En términos generales, es bastante habitual que antes de que una víctima de un ataque Drive-by-Download entre en contacto con el vector de infección se produzcan numerosas redirecciones. Esto quiere decir que antes de llegar al destino, la víctima dará sucesivos saltos, con los consecuentes cambios de dirección, lo que dificulta enormemente la localización del atacante así como la notificación de que se está sufriendo un ataque a las diferentes partes involucradas. Además, su viaje permite al atacante recopilar algunos datos (navegador, versión, plugins instalados…) que pueden ser de utilidad para detectar vulnerabilidades posteriormente y/o decidir si llevar a cabo o no el ataque.

1.1.2.2. Ofuscación

El código JavaScript que se inyecta para llevar a cabo un ataque Drive-by-Download suele estar fuertemente ofuscado. El objetivo no es otro que ocultar el código malicioso para evitar que este sea detectado. La manera más eficaz de lograrlo es utilizar la mayor variedad de técnicas de ofuscación posibles

[2] Estudio en (Marco Cova, Christopher Kruegel, and Giovanni Vigna)

combinándolas en distintas capas.

1.1.2.3. Preparación del ataque

Es conocido que la forma más habitual de actuar a la hora de explotar una vulnerabilidad se centra en la corrupción de la memoria del navegador objetivo. Desde el punto de vista del atacante, para que el plan llegue a buen puerto lo primero es inyectar en la memoria del proceso que controla al navegador el código a ejecutar (Shellcode), para después, aprovechando la explotación de alguna vulnerabilidad del navegador, secuestrar este proceso y ejecutar dichas instrucciones. Es necesario no originar ningún error ya que daría al traste con todo el ataque.

1.1.2.4. Explotación

Una vez tomado el control del navegador llega el momento del verdadero ataque, la instalación del malware que realizará la acción para la que haya sido programado. Los plugins como ActiveX o Flash suelen ser utilizados como puente para dar este último paso.

1.1.2.5. Uso de técnicas combinadas

Para el correcto desarrollo de las cuatro fases anteriores, un atacante debe hacer uso preciso de varias técnicas, algunas de las cuales, tal y como hemos comentado con la ofuscación, no identifican por sí solas un comportamiento sospechoso. Se exime, por tanto, de ser señalados como sitios web comprometidos a aquellos que utilicen alguna de las técnicas, siendo por otro lado claramente culpables los que combinan el uso de varias, dejando claro su objetivo final.

1.1.3. FACTORES QUE HACEN LOS ATAQUES DRIVE-BY-DOWNLOAD EFECTIVOS

Anteriormente se ha mencionado que una de las razones por las que los

ciberdelincuentes habían empezado a usar de manera mucho más frecuente la técnica Drive-by-Download en los últimos diez años era por su mayor efectividad frente a otras alternativas, pero no indicábamos qué tienen de especial estos ataques que los hacen tan peligrosos. Aunque podrían añadirse algunos más, estos son los cinco factores principales que incrementan el riesgo de un ataque Drive-by-Download:

- Las vulnerabilidades en los clientes web están generalizadas: por desgracia es algo normal, la cifra estimada es del 15% CVE (Common Vulnerabilities and Exposures).
- El 45% de los usuarios usan un navegador desfasado: es habitual, tanto Chrome como Internet Explorer (también ahora Microsoft Edge), Mozilla Firefox, Ópera, Safari o cualquier otro navegador entre los más utilizados, se actualizan cada poco tiempo y no todos los usuarios siguen el ritmo, especialmente aquellos con menos conocimiento en este ámbito. Como sucede con cualquier otro software, los equipos al cargo de estas aplicaciones van encontrando y solucionado errores, eliminando los agujeros de seguridad por los que se cuelan los ciberdelincuentes. Mantener tu equipo al día es fundamental para evitar que esos fallos conocidos sean explotados.
- Es necesaria mucha documentación para llevar a cabo estos ataques y por tanto, también es necesario investigar mucha información para obtener datos valiosos que permitan detectar un posible atacante.
- Facilidad para ofuscar el código y que sea entregado al usuario: la ofuscación de código es una conducta bastante habitual y no siempre está ligada con un comportamiento sospechoso. Es bastante normal que muchos sitios webs utilicen estos mecanismos para proteger la integridad de la información que circula por los mismos, lo que supone una oportunidad de oro para que los ciberdelincuentes cuelen su código ofuscado sin saltar ninguna señal de alarma.

- El desconocimiento del mundo online: ¿quién no utiliza Internet para algo a día de hoy? El uso extendido de Internet conlleva que no todo el mundo que se pone delante de un teclado tiene un conocimiento mínimo sobre cómo funciona, sus mecanismos, las posibles trampas que se va a encontrar y como evitarlas o protegerse contra ellas.

1.1.4. IMPACTO DE UN ATAQUE DRIVE-BY-DOWNLOAD

Está bien, ya sabemos del peligro que suponen para cualquier usuario los ataques Drive-by-Download, varios factores, algunos de los cuales escapan por completo al control del usuario que entra en Internet, se unen para hacer de esta técnica un riesgo real. Pero, si acabo siendo una víctima, ¿qué ocurre? La respuesta sencilla sería: nada bueno. Pero como casi siempre cuando hablamos de seguridad web, el tema es bastante más complejo y dependerá en gran medida de las intenciones del atacante con el que te topes.

Como vimos en un apartado anterior, el último paso de un ataque Drive-by-Download considerado exitoso es la descarga, instalación y ejecución del código malicioso. Un malware que se introduce en el sistema causando diversas alteraciones en función de su tipo, normalmente ligado al objetivo con el que ha sido diseñado. Aunque, como decimos, son muchos y muy variados, los principales cambios que se observan tras un ataque Drive-by-Download son:

- Cambios en el registro: normalmente se cambian algunas claves para ganar los privilegios necesarios para realizar las operaciones necesarias, aunque también se modifican y borran otras entradas del registro, lo que puede afectar de manera negativa a su rendimiento y en el peor de los casos, causar daños graves.
- Procesos en ejecución: cuando un equipo resulta infectado después de un ataque Drive-by-Download, el número de procesos en ejecución aumenta de forma notable, llegando incluso a sobrepasar la capacidad del procesador.
- Sustitución de archivos: otra de las estrategias habituales es la

sustitución de algunos archivos clave para el sistema operativo, lo que permite la manipulación en gran medida de su comportamiento.

- Trafico de red: también se observará un incremento brutal del tráfico de red como consecuencia de las conexiones continuas con la web maliciosa y las descargas que se inician tras la infección.
- Incremento paquetes TCP y UDP: el atacante a menudo intentará infectar otros equipos cercanos, por lo es previsible que aparezcan numerosos paquetes TCP y UDP enviados que puedan identificar puertos abiertos y así explorar la posibilidad de expandirse a otros sistemas conectados a la misma red LAN.

Pero no nos olvidemos del que posiblemente sea el mayor impacto de todos, el económico. La 'Figura 3: Infografía ataques Drive-by-Download' muestra una infografía[3], con datos de 2013, que reafirma la peligrosidad de los ataques Drive-by-Download, capaces de llevarse a cabo en medio segundo y de reportar hasta 50.000 dólares diarios a quienes los realizan para un coste total superior a los 5 millones de dólares.

1.2. OFUSCACIÓN

De entre todas las características que componen un ataque Drive-by-Download, este proyecto se centra en la parte de ofuscación de código. Es por este motivo que la presente sección estará dedicada a profundizar más en los diferentes aspectos que rodean esta técnica clave para los ciberdelincuentes que quieren poner en práctica un ataque exitoso.

En primer lugar, vamos a definir ofuscar. Según la RAE puede significar.

1) Deslumbrar, turbar la vista.
2) Oscurecer y hacer sombra.

[3] Infografía extraída de (Sophos.com, 2014)

3) Trastornar, conturbar o confundir las ideas, alucinar.

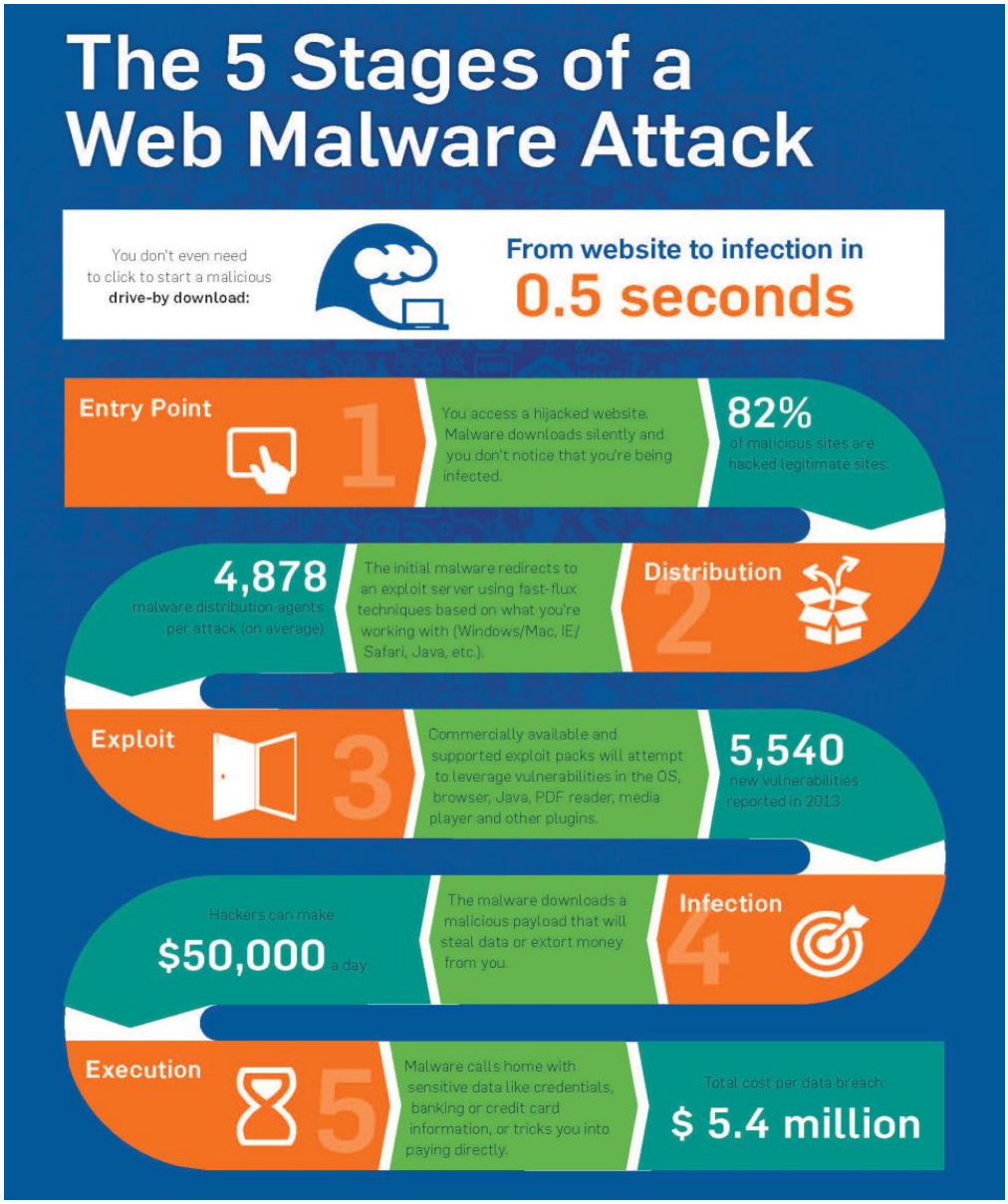

Figura 3: Infografía ataques Drive-by-Download

Aunque ninguna de estas definiciones corresponde exactamente con la que se aplica en nuestro campo, sí que pueden servir para hacerse una idea aproximada, especialmente la tercera entrada. La ofuscación hace alusión al acto intencionado de modificar un código con el objetivo de que este sea difícil de leer e interpretar a simple vista. En nuestro caso aplicado al código JavaScript en el que se ocultan algunos comandos necesarios para que el ataque Drive-by-Download siga su curso.

Esta modificación suele confeccionarse con las propias operaciones primitivas de JavaScript y bien realizada, transforma cualquier código en una

sopa de caracteres imposible de descifrar manualmente. Se puede comprobar perfectamente en la ‘Figura 4: Ejemplo código ofuscado’, donde se muestra un código bastante sencillo (izquierda), y su equivalente ofuscado (derecha).

```
function NewObject(prefix)
{
   var count=0;
   this.SayHello=function(msg)
   {
      count++;
      alert(prefix+msg);
   }
   this.GetCount=function()
   {
      return count;
   }
}
var obj=new NewObject("Message : ");
obj.SayHello("You are welcome.");
```

```
var _0x40e2=
["\x53\x61\x79\x48\x65\x6C\x6C\x6F","\x47\x65\x74\x43\x6F\x75\x6E\x74","\x4D\x65\x73\x73\x61\x67\x65\x20\x3A\x20","\x59\x6F\x75\x20\x61\x72\x65\x20\x77\x65\x6C\x63\x6F\x6D\x65\x2E"];function NewObject(_0xdfa3x2){var _0xdfa3x3=0;this[_0x40e2[0]]= function(_0xdfa3x4){_0xdfa3x3++;alert(_0xdfa3x2+ _0xdfa3x4)};this[_0x40e2[1]]= function(){return _0xdfa3x3}}var obj= new NewObject(_0x40e2[2]);obj.SayHello(_0x40e2[3])
```

Figura 4: Ejemplo código ofuscado

1.2.1. TÉCNICAS DE OFUSCACIÓN

Obviamente, el resultado que se alcanza en el ejemplo anterior no se obtiene aplicando una simple modificación, sino que es consecuencia de una transformación masiva y sistemática del código original. Las técnicas de ofuscación se pueden categorizar en función de las operaciones que realizan, y son cuatro las principales categorías que se encuentran.

1.2.1.1. Ofuscación aleatoria

Insertar o cambiar algunos elementos de los códigos de JavaScript sin que esto afecte la semántica de los mismos. Aunque sencillas, estas técnicas son bastante útiles para dificultar la lectura de los códigos ofuscados.

- Espacios en blanco: se introducen tanto espacios como líneas, tabulaciones, entre otros.

  Ejemplo:

No ofuscado	**Ofuscado**

<script type="text/javascript">
document.write("Soy ET");
alert("Hola querido terrestre");
</script>
	<script type= "text/javascript">
	document.write
	("Soy ET") ;
	alert("Hola querido terrestre");
	</script>

- Comentarios: inserta comentarios aleatoriamente.

  Ejemplo:

No ofuscado	**Ofuscado**

<script type="text/javascript">
document.write("Soy ET");
alert("Hola querido terrestre");
</script>
	<script type="text/javascript">
	//gZRnOsS11eXaumf
	document.write("Soy ET")/*w2uyhES
	yNT0x2ZNwJVqKb39y9liUeWQpLTC
	XuMVkadDhQ8PX14or*/;
	alert("Hola querido terrestre");
	//t6jfRtzG3nmPly4
	</script>

- Nombres de variables y funciones: se sustituyen por nombres aleatorios o nombres no identificativos, por consiguiente, su identificación se vuelve una ardua tarea.

  Ejemplo:

No ofuscado	Ofuscado
nombre = “Lucas”;	2sd46wd562erw = base64_decode("THVjYXMNCg==”);

De estas tres, las dos primeras son ignoradas por los intérpretes de Javascrip, no ocurre así con la tercera, por lo que su aplicación se puede considerar casi obligatoria si se quiere obtener un resultado apropiado.

1.2.1.2. Ofuscación de datos

Convierte una variable o una constante en los resultados de cálculo de una o varias variables o constantes. O lo que es lo mismo, logra que la mera representación de un dato sea mucho más compleja.

- División String (String Splitting): convierte un string en la concatenación de varios strings, le pueden usar diferentes métodos para lograr esto. Para complicarlo aún más, pueden cambiar el orden y asignarles variables a los strings.

  Ejemplo:

No ofuscado	Ofuscado

function clave(){
  alert('Hello World');
}
eval(clave());
	var string1=“cla”;
	var string2=“ve”;
	var string3=”()”;
	function clave(){
	  alert('Hello World');
	}
	eval(string1+string2.concat(string3));

- Sustitución palabras clave (Keywords Substitution): usar variables

en lugar de palabras clave. Nuevamente, el propósito es entorpecer la búsqueda e identificación de dichas palabras.

Ejemplo:

No ofuscado	**Ofuscado**
alert(“PalabraClave”);	var str = “PalabraClave”; var res = str.replace(“PalabraClave”, ”OtraPalabra”); alert(res);

- Números: pasan a estar representados como operaciones. Por ejemplo, un número 10 podría aparecer como 11-1, 5*2, etc. Utilizada de forma repetitiva, esta técnica supone una barrera prácticamente infranqueable si queremos obtener el resultado de alguna función/operación de forma rápida.

Ejemplo:

No ofuscado	**Ofuscado**

var x=10;
var y=20;
var res=x+y;
	var x=eval(“2*5”);
	var y=30-10;
	var res=x+y;

1.2.1.3. Ofuscación por codificación

Aunque las técnicas anteriores pueden resultar de gran utilidad, su aplicación por sí sola no serviría para lograr un resultado óptimo. No obstante, se suelen acompañar del uso de técnicas más complejas.

- Cambio de codificación: se trata de convertir el código “normal” en caracteres ASCII, Unicode o hexadecimal.

Ejemplo:

No ofuscado	**Ofuscado**
nombre=“Lucas”;	nombre=”\x4C\x75\x63\x61\x73”;

- Funciones de codificación personalizadas: los atacantes generalmente utilizan una función de codificación para crear el código ofuscado y una función de decodificación para decodificarlo durante la ejecución, diseñadas por ellos mismos, de tal forma que cualquier otro no sepa lo que verdaderamente se está haciendo en esas líneas.

  Ejemplo[4]:

No ofuscado	**Ofuscado**

document.write('Hello world!');	function decode(c){
	var mystring="";
	var i =0;
	for(var i=0;i <c.length;i++){
	mystring=mystring +
	String.fromCharCode(c.charCodeAt(i)-1);
	}
	return mystring;
	}
	var c=”epdvnfou/xsjuf)(Ifmmp!xpsme(*”;
	var mystring=decode(c);
	eval(mystring);

Para que este código ofuscado con la función para decodificar el JavaScript

[4] Este y otros ejemplos en (Wei Xu, Fangfang Zhang and Sencun Zhu)

tenga sentido y funcione, se ha de implementar una función de codificación que se aplique previamente al código original no ofuscado:

Función de codificación

```
function encode(mystring){
  var c="";
  var i =0;
  for(var i=0;i<mystring.length;i++){
    c=c+String.fromCharCode(mystring.charCodeAt(i)+1);
  }
return c
}
var c=encode("document.write('Hello world!')");
document.write(c);
```

- Métodos estándar de codificación y decodificación: los métodos propios pueden combinarse con otros estándares de cifrado y descifrado como puede ser el JScript.Encode, diseñado por Microsoft para codificar código JavaScript.

1.2.1.4. Ofuscación estructura lógica

Este tipo de técnica de ofuscación se puede resumir como la alteración del orden de ejecución de las instrucciones en un código JavaScript, modificando la estructura lógica, pero sin afectar a la semántica original.

- Inserción de instrucciones: se añaden instrucciones independientes de la funcionalidad que se está implementando.

  Ejemplo:

No ofuscado	Ofuscado

document.write('Hello world');	var i=111;
	i=i+1;
	if(i<10){
	alert("Warning");
	}
	document.write('Hello world');
	i=i+1;

- Añadir o cambiar condicionales: aparecen condicionales (if, else, switch, for, while) adicionales y otras son manipuladas.

  Ejemplo:

No ofuscado	Ofuscado

document.write('Hello world');	var i=0;
	for(i=0;i<=10000;i++){
	if(i==1){
	document.write('Hello world');
	}
	}

1.2.2. USO DE LA OFUSCACIÓN

La ofuscación del código es una técnica muy usada para la realización de ataques Drive-by-Download, pero como se ha comentado en apartados anteriores, no solo los malhechores utilizan esta técnica para embarullar su

código y hacerlo ilegible. Es decir, su uso está extendido más allá del ámbito criminal, pero ¿hasta qué punto se utiliza la ofuscación?

Lo primero que se ha de tener en cuenta es que actualmente, no existe una herramienta o software capaz de detectar de forma precisa si un código está ofuscado, menos aún de identificar cuáles de las técnicas/categorías han sido utilizadas. Para obtener datos concretos y mínimamente relevantes se ha de examinar minuciosamente y de forma manual una considerable cantidad de ejemplos, tal y como han hecho algunos estudios.

La gráfica de la 'Figura 5: Uso de técnicas de ofuscación por categorías', extraída de uno de estos estudios[5], refleja el número de categorías de ofuscación que aparecen en una muestra relativamente amplia de códigos. Cómo se puede apreciar, aproximadamente el 70% de los códigos analizados han sido modificados con alguna de las técnicas de ofuscación descritas anteriormente. De entre todas, la división de strings, la sustitución de palabras clave, la codificación con caracteres ASCII, Unicode o hexadecimal y el uso de funciones de codificación personalizadas, son las más utilizadas.

Resulta interesante observar cómo una fracción considerable de los códigos, un 21%, tiene aplicadas técnicas enmarcadas dentro de dos categorías diferentes pero solo en algunos aparecen técnicas de tres (7%) o cuatro categorías (2%).

1.2.1. ¿SON EFECTIVOS LOS ANTIVIRUS?

Si un 70% de los códigos JavaScript han sido modificados con al menos una de las técnicas de ofuscación conocidas, como usuarios es fácil caer en la suposición de que las principales herramientas de seguridad que utilizamos para proteger nuestros equipos, los antivirus, estarán preparados para hacer frente a esta circunstancia.

[5] Estudio en (Wei Xu, Fangfang Zhang and Sencun Zhu)

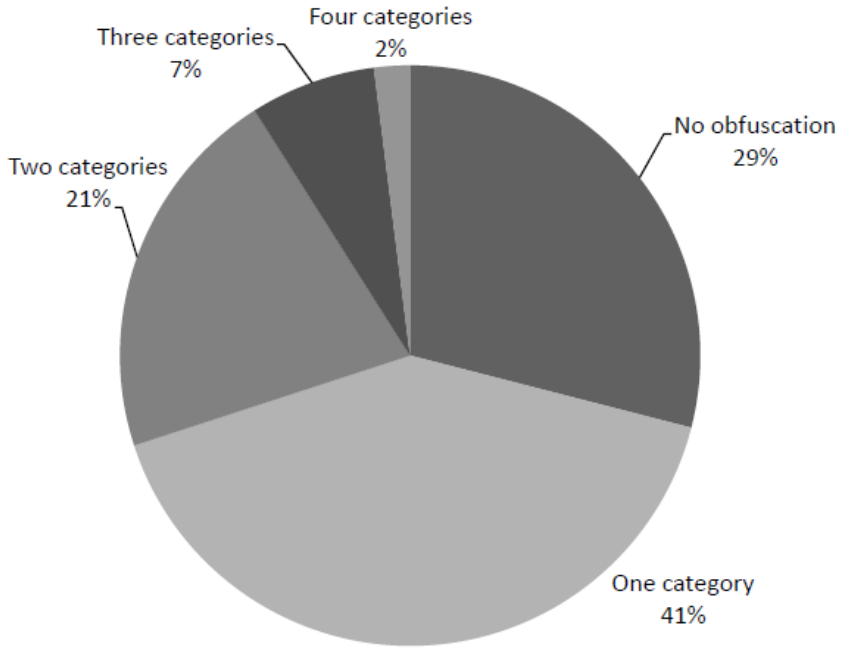

Figura 5: Uso de técnicas de ofuscación por categorías

La realidad es que no. Es cierto que la mayoría de antivirus son capaces de responder con bastante solvencia a ataques realizados con código sin ofuscar, pero es igual de cierto que sus estadísticas de efectividad caen en picado cuando se aplican algunas las técnicas presentadas en apartados anteriores. Incluso las más sencillas dificultan la labor de estos programas.

El mismo estudio al que se hacía referencia anteriormente establece que la media de detecciones de 20 de los antivirus más utilizados (considerados muchos de ellos como los mejores en su ámbito) se sitúa cercana al 90% cuando el código malicioso se presenta sin ofuscación. Pero baja hasta prácticamente la mitad cuando se modifica utilizando algunas técnicas de ofuscación.

De este modo, el número positivos es de solo el 55% cuando se aplican técnicas de la categoría que hemos llamado ofuscación aleatoria, y que como se ha explicado, se basan normalmente en operaciones sencillas, a priori fáciles de ignorar y muy sencillas de implementar (la 'Figura 6: Detección de código malicioso con y sin ofuscación aleatoria' muestra el bajón de efectividad en prácticamente todos los programas antivirus).

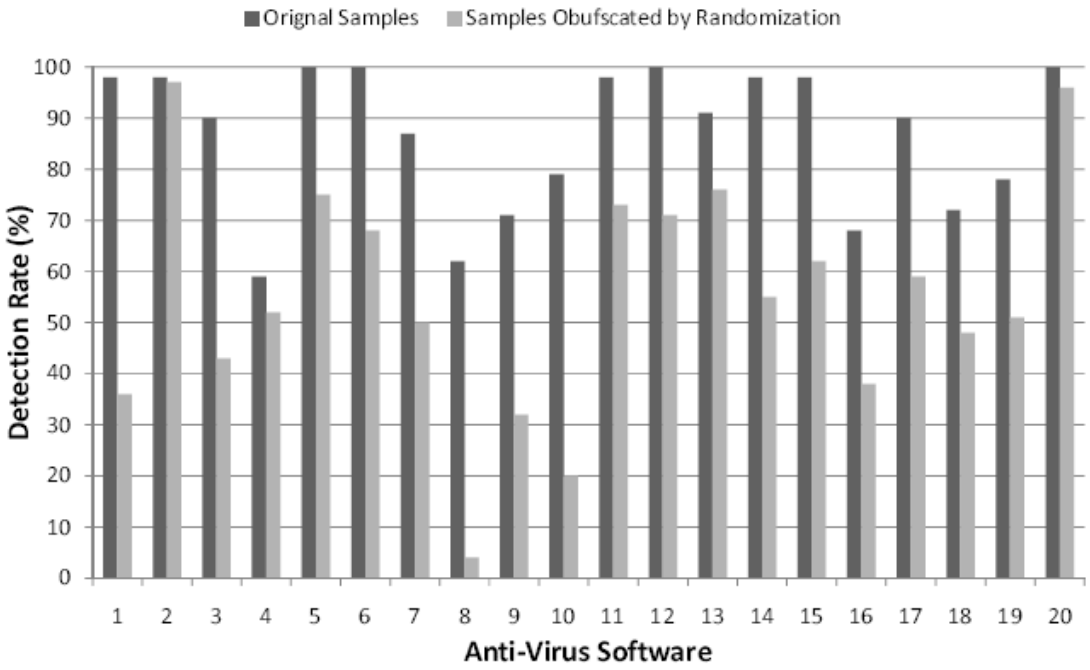

Figura 6: Detección de código malicioso con y sin ofuscación aleatoria

La cosa no hace sino empeorar cuando incorporamos ofuscación de datos, reduciéndose la efectividad media de estos 20 antivirus a un raquítico 45%. Pero aquí no queda todo, si se incluyen técnicas de ofuscación por codificación, prácticamente ninguno de los software utilizados es capaz de detectar que el código es malicioso. Esto se debe a su funcionamiento, basado normalmente en la detección de firmas digitales, una marca única que se compara con una base de datos de virus en busca de coincidencias.

En definitiva, el uso de la ofuscación por parte de los atacantes está plenamente justificado por la capacidad que les otorga esta técnica para evadir con gran efectividad los sistemas de detección de prácticamente la totalidad de los programas antivirus disponibles.

1.3. OBJETIVOS DEL PROYECTO

A continuación se definirán cuáles son los objetivos concretos establecidos para el proyecto y qué papel jugaría la característica que se ha implementado

como “peón” dentro de un sistema mucho más complejo que teóricamente sería capaz de detectar y detener un ataque Drive-by-Download con suma efectividad.

En el apartado 1.2.1. respondíamos a la pregunta ¿son efectivos los antivirus frente a código malicioso ofuscado? La respuesta es rotunda, no. Incluso las técnicas más sencillas de ofuscación, combinadas, permiten a un atacante saltarse los sistemas de detección de los mejores programas antivirus una de cada dos veces que se intenta un ataque. Está claro que hay mucho trabajo por hacer en este sentido.

Este proyecto no pretende suplir esta carencia de los antivirus, programas que tienen cantidad de grandes profesionales detrás trabajando para mejorar la seguridad de los usuarios de Internet. Lo que intentamos es establecer la base de un sistema que sea capaz de dictaminar si un código está o no ofuscado con precisión, una primera piedra a partir de la cual se podría construir un muro (equivaldría a un sistema completo que incorporase todas las características descritas en el apartado 1.1.2.) que taponase la autopista que ahora mismo hay abierta entre los ciberdelincuentes y nuestros equipos como consecuencia de uso de técnicas de ofuscación y su facilidad para evadir los mecanismos de detección de los programas antivirus.

Para que este sistema maximizase su efectividad contra ataques Drive-by-Download debería actuar desde el propio navegador, poniendo en alerta a todo el equipo en caso de detectar patrones sospechosos al visitar un sitio web. Es por este motivo que utilizaremos HTMLUnit, un navegador web sin interfaz que nos permite diseñar su comportamiento para establecer una herramienta que extraiga la información necesaria para decidir un instante después de descargar una página web si el código JavaScript está o no ofuscado.

El primer paso será por tanto descargar la página y extraer todo el código JavaScript que se ejecute directamente o con una llamada a otro servidor. Una vez realizada este proceso deberemos analizar el código y explorar si se han aplicado las técnicas conocidas de ofuscación. A priori puede parecer sencillo apreciar si un código está o no ofuscado, conocemos cuáles son las técnicas y muchas de ellas son fáciles de localizar en un vistazo. Pero el ojo engaña, y algunas técnicas utilizadas por los responsables de las web para comprimir el código o hacerlo de difícil lectura para alguien ajeno, pueden dar la falsa

sensación de encontrarse aunque un código ofuscado.

Para decidir si el código está o no ofuscado será necesario realizar un entrenamiento, haciendo uso de varios ejemplos de código JavaScript y diversas herramientas gratuitas de ofuscación de código que nos permitirán establecer cuáles son los niveles “normales” de aparición de algunas características relacionadas directamente con el uso de diferentes técnicas de ofuscación.

Con todo, mostraremos una interfaz donde se puede introducir la dirección de la web que queremos comprobar, y automáticamente tendremos un resultado positivo, en caso de concluir que dicha web contiene código ofuscado y por tanto sería susceptible de haber sido comprometida (aunque no necesariamente como hemos dejado claro en varias ocasiones), o negativo, si no se observan evidencias suficientes.

2. ANÁLISIS Y METODOLOGÍA SEGUIDA EN EL PROYECTO

En los capítulos anteriores hemos sentado las bases teóricas del problema que queremos afrontar en este proyecto y establecido las metas que se persiguen con su realización. A continuación se van a exponer tanto la planificación estimada para la realización del proyecto como los requisitos (funcionales y no funcionales) que se han de satisfacer para considerar su realización como exitosa, la metodología seguida y un estudio económico donde se recojan los materiales utilizados y los costes que irían asociados a la ejecución y resolución del mismo.

2.1. REQUISITOS

En este apartado se van a establecer los requisitos del sistema a desarrollar, los cuales permiten medir las necesidades y el grado en el que han de ser satisfechas con el que podríamos considerar que la realización del proyecto ha finalizado de forma exitosa. Se diferenciará entre requisitos funcionales y no funcionales.

2.1.1. REQUISITOS FUNCIONALES

Este punto hace referencia a los requisitos funcionales, aquellos que indican cuál debe ser el comportamiento del sistema ante diferentes condiciones con el fin de que se puedan alcanzar los objetivos del proyecto.

Los requisitos funcionales son:

- El sistema debe ser capaz de descargar diferentes páginas web emulando distintos navegadores.
- El sistema ha de extraer todo el código JavaScript ejecutado al descargar una web, independientemente de que se encuentre de forma íntegra en el documento HTML o se realice una llamada externa mediante una URL.

- El sistema ha de buscar un mínimo de características relacionadas al código ofuscado que garanticen un funcionamiento óptimo de la herramienta implementada.
- El entrenamiento ha de llevarse a cabo con el número suficiente de muestras y herramientas de ofuscación de modo que los datos puedan ser representativos de un conjunto mucho mayor.
- Se ha de diseñar un clasificador probabilístico que haga uso de los datos obtenidos en el entrenamiento.
- El sistema debe disponer de una interfaz que facilite la introducción y presentación de la información.
- El sistema ha de responder en un solo paso: el usuario que haga uso de la misma solo tendrá que introducir la dirección de la web a analizar y se devolverá una respuesta sobre si contiene o no código JavaScript ofuscado. Opcionalmente se pueden mostrar los resultados del análisis a fin de facilitar la comparativa de los mismos.
- El sistema debe ofrecer siempre una respuesta sobre la ofuscación del código, a ser posible de forma visual en la interfaz. SI en caso de que se encuentren los indicios suficientes y NO en caso contrario.

2.1.2. REQUISITOS NO FUNCIONALES

Seguidamente se definen los requisitos no funcionales, aquellos que no están ligados directamente al comportamiento del sistema sino que permiten establecer un marco de accesibilidad para la introducción de posibles mejoras en el futuro, la modificación de ciertas características, el mantenimiento y actualización del mismo, etcétera.

Los requisitos no funcionales son:

- El código debe ser extensible con el objetivo de facilitar la introducción de futuras mejoras.
- Las características del código ofuscado se han de implementar de

forma individual de tal forma que cualquiera pueda modificarlas o introducir alguna nueva sin afectar al resto.

- El sistema debe estar preparado para su incorporación a una herramienta de nivel superior que incluya la detección de todas las fases de un ataque Drive-by-Download (o varias de ellas).
- Se han de utilizar las bibliotecas de HTMLUnit sin modificaciones, ya que de lo contrario, se dificultaría la labor de los terceros que desearan hacer algún cambio.
- El sistema debe estar programado en un lenguaje que funcione de manera eficiente.
- El clasificador debe implementarse de forma separada al resto de módulos que integran el sistema. Esto facilitará la integración de cualquier otro clasificador probabilístico que sea capaz de ofrecer mejores, o al menos distintos, resultados.

2.2. PLANIFICACIÓN

En este punto disponemos, de forma aproximada, la planificación seguida para el desarrollo del proyecto, desde que comenzamos con el estudio de las herramientas y tecnologías que íbamos a utilizar hasta su conclusión con las pruebas pertinentes. Su complejidad obliga a dividir el trabajo en tareas que hemos agrupado en siete fases diferentes, las cuales se resumen en la siguiente tabla, aunque serán explicadas posteriormente una a una.

Fase	Tarea	Fecha de inicio	Fecha de finalización
1	Revisión del estado del arte	15/02/2016	28/02/2016
2	Aprendiendo a utilizar HTMLUnit	29/02/2016	20/03/2016
3	Implementación (1)	21/03/2016	24/04/2016
4	Entrenamiento	25/04/2016	01/05/2016
5	Análisis probabilístico de los datos	02/05/2016	08/05/2016
6	Implementación (2)	09/05/2016	25/05/2016
7	Prueba y evaluación	26/05/2016	29/05/2016

Tabla 1: Resumen planificación

2.2.1. VISIÓN GLOBAL

La siguiente ‘Figura 7: Planificación’ muestra de manera más detallada y en forma de diagrama todas las tareas en las que se ha dividido el trabajo, con las fechas aproximadas en las que han sido realizadas.

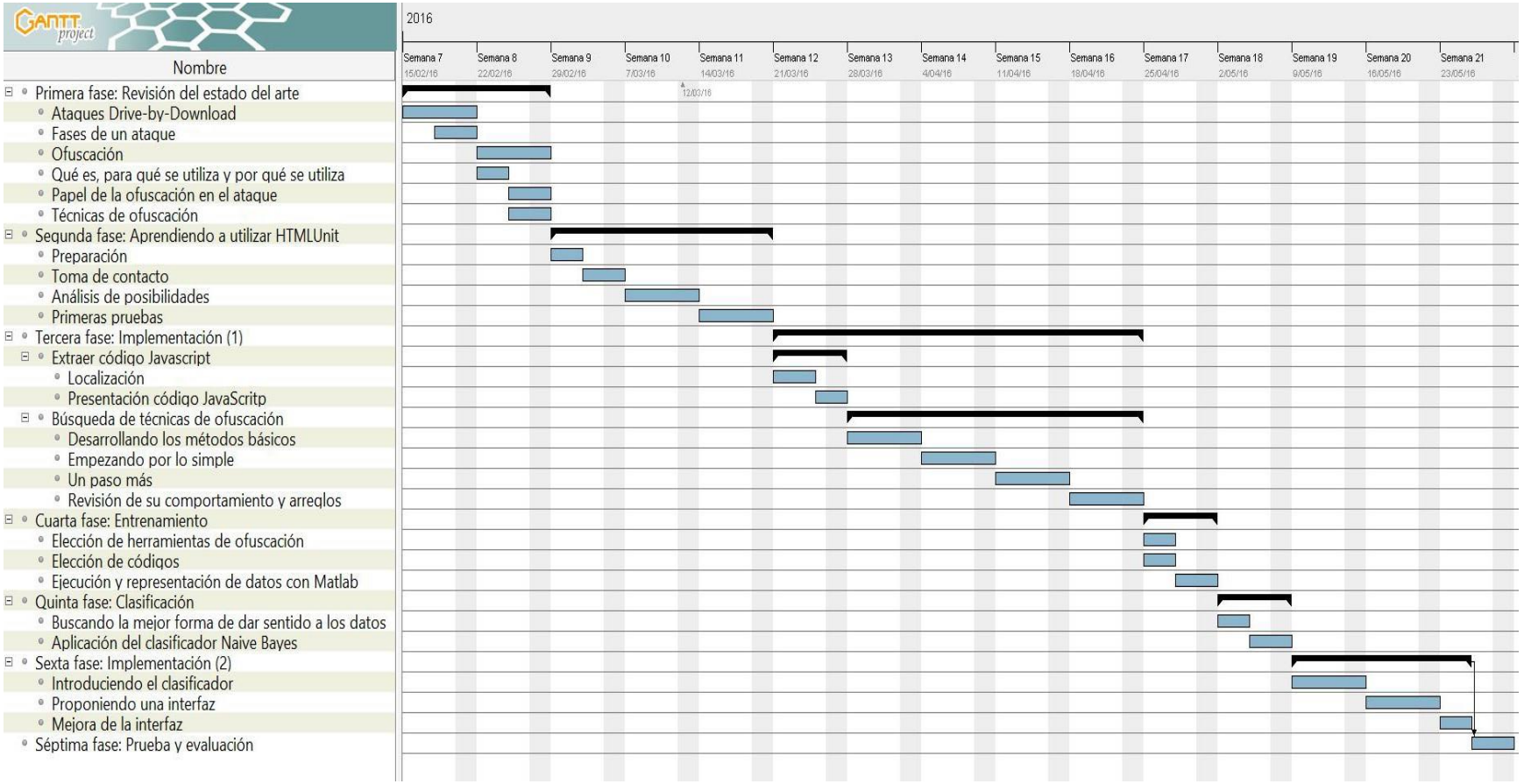

Figura 7: Planificación

2.2.2. FASE 1: REVISIÓN DEL ESTADO DEL ARTE

Figura 8: Primera fase planificación

Antes de empezar a trabajar se necesita realizar un estudio exhaustivo de las diferentes partes implicadas en el proyecto. Comenzando, cómo no, por los ataques Drive-by-Download. Es importante interiorizar cómo funcionan, cuáles son las fases en las que se divide su acción y tener claro donde encaja nuestro proyecto en este esquema antes de avanzar. También es aconsejable estudiar la relevancia de este tipo de ataques, tan utilizado en los últimos años por los ciberdelincuentes, y tan peligroso para los usuarios que navegan por Internet.

Una vez recabada toda la información posible, consultados los múltiples estudios existentes sobre los ataques Drive-by-Download y ordenados los conceptos, pasamos a empaparnos con lo que realmente concierne al proyecto, la ofuscación. Nuevamente, es importante tener claro qué es, para qué se utiliza y por qué se utiliza de manera tan generalizada. Asimilada la idea, no resulta fácil para personas que están fuera del sector, de la ofuscación, se ha de profundizar en las técnicas existentes, prestando atención a aquellas que son más utilizadas, planteando desde ya posibles maneras de implementar un detector de código ofuscado con estas técnicas.

2.2.3. FASE 2: APRENDIENDO A UTILIZAR HTMLUNIT

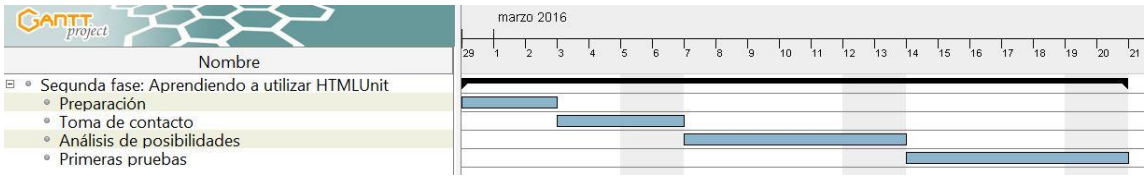

Figura 9: Segunda fase planificación

Una vez se tiene claro lo que se quiere hacer, se ha de seguir aprendiendo, esta vez a manejar las herramientas y tecnologías implicadas. El desarrollo de este proyecto gira entorno a HTMLUnit, el software creado por Mike Bowler viene como anillo al dedo para cumplir con el propósito marcado, pero dar los primeros pasos con él no es del todo fácil.

Lo primero será instalar las librerías en uno de los programas de edición de Java, se aconseja el uso de Eclipse, aunque podría valer cualquier otro. Para la toma de contacto inicial se aconseja la puesta en marcha de alguno de los muchos ejemplos que se pueden encontrar en la página web de HTMLUnit. Esto nos permitirá tener una idea mucho más clara de cómo funciona un navegador sin interfaz.

Seguidamente, hemos de pasar a la acción. Esto implica en primera instancia el repaso y análisis detallado de todas las clases, objetos y métodos que incluyen las librerías de HTMLUnit. Algunos de estos elementos nos servirán para implementar el sistema que buscamos, y elegir bien entre todas las posibilidades (no son pocas) es clave para no entorpecer la marcha del proyecto.

Para asegurarse de que se toma el camino correcto, es primordial la realización de unas pruebas sencillas, no es necesario complicarse a estas alturas, que nos aporten no solo cierta experiencia en el uso de HTMLUnit, sino también una valiosa información sobre el comportamiento de algunos de sus elementos que podrá ser aprovechada posteriormente.

2.2.4. FASE 3: IMPLEMENTACIÓN (1)

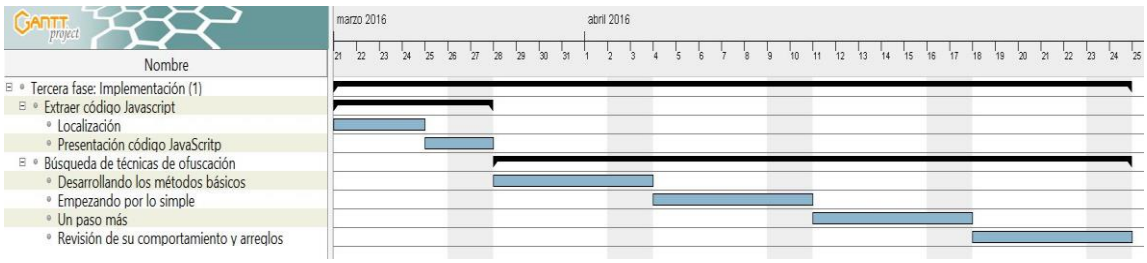

Figura 10: Tercera fase planificación

Comienza la implementación. Estableciendo un símil con una carrera, la primera meta volante en esta fase estaría situada en la localización y

presentación del código JavaScript que ejecuta una página web al ser descargada. Básicamente se ha de extraer el contenido de las etiquetas “script” del documento HTML, estableciendo conexión con la URL indicada (campo “src”) en caso de no encontrar el código JavaScript de forma íntegra en este documento.

Puede resultar de gran utilidad la escritura de un escueto documento HTML que permita comprobar que lo implementado funciona correctamente antes de pasar a probarlo con webs ajenas de mayor tamaño. Por último, se tendría que imprimir todo el código JavaScript localizado en un documento de texto aparte.

Una vez se tiene el código JavaScript al completo en un documento de texto, se ha de realizar un barrido en búsqueda de características que indiquen la existencia de código ofuscado o no. Es de gran valor la creación de métodos básicos como un buscador de palabras o un contador de caracteres. Si los métodos funcionan correctamente, serán de enorme ayuda a la hora de implementar la búsqueda de características, especialmente las básicas (mínimas necesarias para que el proyecto tenga validez). Sobre este mínimo de características se añaden cuántas más características adicionales mejor.

2.2.5. FASE 4: ENTRENAMIENTO

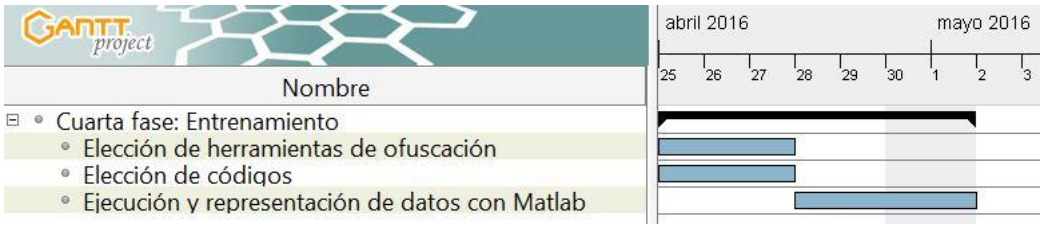

Figura 11: Cuarta fase planificación

Una vez se es capaz de extraer un número interesante de características que ayuden a diferenciar entre códigos ofuscados y códigos no ofuscados, toca poner en marcha nuestro sistema para comprobar su validez. Para tal propósito, se plantea la ejecución de un entrenamiento en el que se haga uso de 3 a 5 herramientas de ofuscación diferentes y entre 5 y 10 muestras de códigos JavaScript.

Se trata de pasar por el sistema implementado las muestras sin ofuscar y posteriormente ofuscadas con cada una de las herramientas.

2.2.6. FASE 5: CLASIFICACIÓN

Figura 12: Quinta fase planificación

Con los datos y gráficas extraídos del entrenamiento, tendremos información suficiente para establecer un clasificador que decida si un código cualquiera está o no ofuscado en base a las características implementadas. Para tal fin se hará uso de un clasificador Naive Bayes.

2.2.7. FASE 6: IMPLEMENTACIÓN (2)

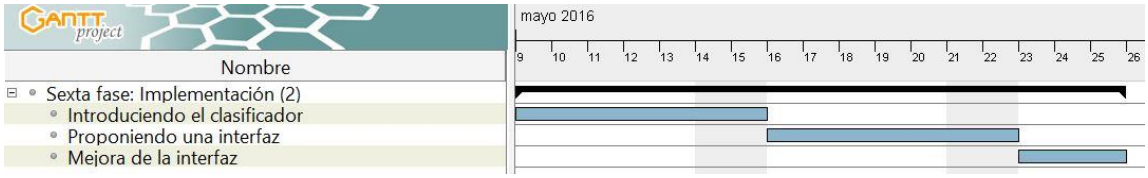

Figura 13: Sexta fase planificación

A continuación, se ha de incorporar el clasificador diseñado al sistema implementado, de tal manera que la toma de decisión sea automática y no requiera de un paso extra.

Para que todo luzca mejor y el uso del sistema sea accesible, es importante diseñar una interfaz donde será posible introducir la dirección del sitio web que se desea analizar y se mostrará los resultados del análisis de características y la decisión del clasificador que indique si el código está o no ofuscado definitivamente.

2.2.8. FASE 7: PRUEBAS

Figura 14: Séptima fase planificación

Por último, llega la hora de poner a prueba el sistema. Se introducirán varias direcciones de webs, y se observarán los resultados obtenidos. Es importante que alguna de estas webs sea conocida y se pueda verificar si el sistema acierta o no al decidir sobre la ofuscación del código JavaScript.

2.3. METODOLOGÍA

La metodología propuesta para el desarrollo satisfactorio del proyecto está íntimamente relacionada con la planificación seguida.

- Repaso de los principales vectores de ataque en la actualidad y estudio en profundidad de los diferentes aspectos de un ataque Drive-by-Download.
- Estudio de la ofuscación con especial atención a las diferentes técnicas que se pueden utilizar.
- Puesta en marcha de HTMLUnit, introducción a su uso e investigación de sus posibilidades.
- Diseño general del sistema que se ha de implementar y los documentos de apoyo necesarios.
- Implementación de la descarga de páginas webs emulando la descarga con diferentes navegadores (Chrome, Firefox, IE…).
- Diseño e implementación del primer módulo del sistema que permite extraer el código JavaScript.

- Diseño e implementación del segundo módulo del sistema, comprobando de forma individual la aportación de cada característica que se incorpora.
- Diseño del campo de entrenamiento de forma que la extracción de los datos necesarios esté lo más automatizada posible.
- Diseño y aplicación del clasificador Naive Bayes.
- Finalización de la implementación y diseño de una interfaz.
- Comprobación del funcionamiento del sistema implementado mediante baterías de pruebas.

2.4. TECNOLOGÍAS UTILIZADAS PARA EL DESARROLLO DEL PROYECTO

Para la realización de este Trabajo Fin de Grado han sido necesarios muchos de los conocimientos adquiridos durante la carrera. El primero y más importante, la habilidad para programar y en concreto, del uso del lenguaje Java con el que se ha implementado toda la herramienta. Aunque no ha sido necesario un conocimiento en profundidad de HTML y JavaScript, sí que han sido de utilidad los conceptos aprendidos en varias asignaturas para comprender su funcionamiento y así, facilitar sobre todo la primera parte del proyecto. No menos importante ha sido la experiencia adquirida con el uso de Matlab, una herramienta que a pesar de servir más como apoyo que como parte fundamental de este proyecto, ha sido de gran ayuda a la hora de agilizar los cálculos, las representaciones y demás operaciones, especialmente durante la parte de entrenamiento.

Pero no todo estaba aprendido. Antes de comenzar ha sido necesario familiarizarse con la herramienta Eclipse, la parte fácil, y sobre todo con HTMLUnit. Este navegador sin interfaz ofrece un sinfín de posibilidades pero su control ha requerido de un intenso estudio previo. Son tantas las cosas que se

pueden hacer con esta herramienta que había que el hecho de encontrar el camino ideal (había bastantes alternativas) para llegar a las metas establecidas ya suponía un reto en sí.

En este apartado se enumerarán las principales tecnologías que han sido utilizadas durante el desarrollo del proyecto. Cada una contendrá una explicación breve de en qué consiste, su papel dentro del proyecto y en caso de ser necesario, información adicional que justifique su elección.

2.4.1. HTMLUnit

HTMLUnit, desarrollado originalmente por Mike Bowler de Gargoyle Software y lanzado en mayo de 2002 bajo la licencia Apache 2, es un navegador sin interfaz escrito en Java capaz de modelar documentos HTML, proporciona una API que le permite descargar páginas, rellenar formularios, pinchar en enlaces y en general, emular todas las partes de un navegador típico. Es compatible con JavaScript y puede trabajar con bibliotecas complejas tales como AJAX, simulando Chrome, Firefox o Internet Explorer dependiendo de la configuración utilizada. Está pensado como una forma de simular un navegador para la realización de pruebas y está destinado realmente a ser utilizado dentro de otro marco de pruebas como JUnit o TestNG.

Figura 15: Logo Gargoyle Software/HTMLUnit

Constituye la herramienta central de nuestro proyecto. El sistema

implementado es un navegador que hace las veces de analizador y clasificador de páginas web en función de la ofuscación del código JavaScript.

2.4.1.1. ¿Por qué HTMLUnit?

Para la realización de este proyecto hemos optado por HTMLUnit como eje central del desarrollo por sus características únicas:

- Permite simular diferentes navegadores y distintas versiones de los mismos.
- Soporte para los protocolos HTTP y HTTPS.
- Soporte para cookies.
- Capacidad para especificar si las respuestas desde el servidor deben lanzar excepciones o por el contrario deben ser devueltas como páginas del tipo apropiado, basado en el tipo de contenido.
- Soporta los métodos POST, GET, HEAD, DELETE…
- Posibilidad de personalizar los encabezados de la solicitud.
- Soporte para HTML que incluye acceso al DOM, formularios, pinchar en los enlaces…
- Soporte para servidor proxy.
- Soporte para autenticación básica y NTLM.
- Excelente soporte para JavaScript.

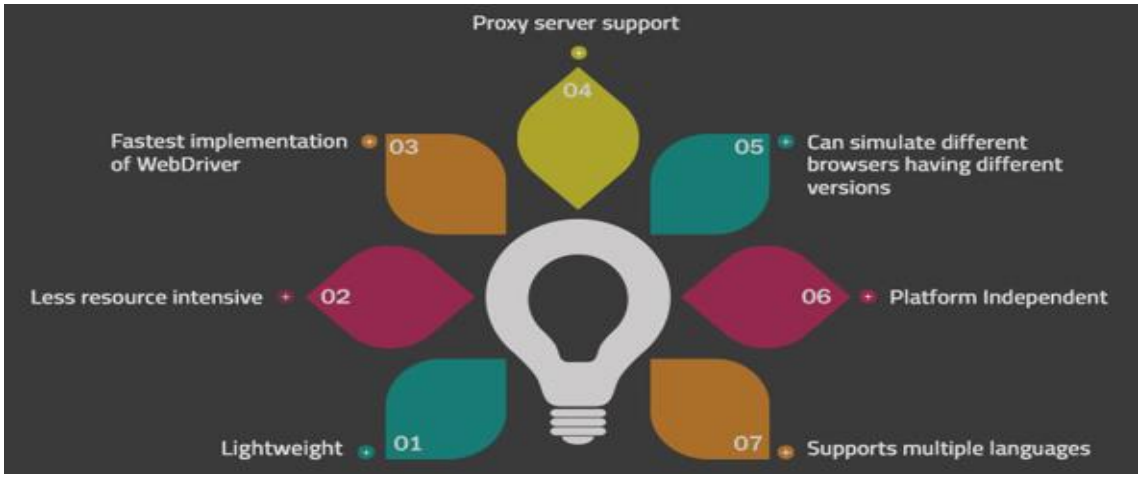

Figura 16: Características HMTLUnit

A continuación se ofrece una pequeña descripción de algunos de los objetos y métodos que se han utilizado entre los muchos disponibles en HTMLUnit, y cuya disponibilidad ha sido crucial a la hora de decantarse por esta herramienta para el desarrollo del proyecto.

- WebClient: Constituye el principal punto de partida en HtmlUnit, esta clase simula un navegador web.
    - BrowserVersion: Los objetos de esta clase representan una versión concreta de un navegador determinado. Se pasa como parámetro a WebClient lo que permite emular diferentes versiones de Google Chrome, Internet Explorer, Mozilla Firefox, etcétera.
    - getPage: Método que permite obtener una página HTML, indicada por su URL, como un objeto HtmlPage.
- HtmlPage: Esta clase representa una página HTML devuelta desde un servidor y proporciona diferentes métodos para acceder a los contenidos de la página.
    - getElementsByTagName: Este método permite obtener en una lista de DomElement con todos aquellos elementos de la página marcados con una etiqueta HTML específica. Sirve para localizar los diferentes scripts.
- DomElement: Representa un elemento del DOM (Document Object Model), o lo que es lo mismo, uno de los objetos que forman el documento HTML.
    - getTextContent(): Con este método se puede acceder al texto contenido en elemento DOM concreto, por ejemplo el código JavaScript de un elemento tipo ‘script’.
    - getAttribute: Método que facilita la información contenida en uno de los atributos del elemento DOM en cuestión. En este caso se muestra especial interés por el ‘src’.

2.4.2. JAVA

Java, desarrollado por James Gosling y publicado en 1995 como un componente fundamental de la plataforma Java de Sun Microsystems, es un lenguaje de programación de propósito general, concurrente, orientado a objetos que fue diseñado específicamente para tener tan pocas dependencias de implementación como fuera posible. Una de sus grandes particularidades es que fue diseñado con la intención de permitir que los desarrolladores de aplicaciones escriban el programa una vez y lo ejecuten en cualquier dispositivo (conocido en inglés como WORA, o "write once, run anywhere"). Desde hace unos años, Java se encuentra entre los lenguajes de programación más populares.

Figura 17: Logo Java

Es el lenguaje que se ha utilizado para la implementación de la herramienta propuesta para este trabajo.

2.4.2.1. ¿Por qué Java?

En relación a la elección de HTMLUnit como herramienta principal, haremos uso de Java como lenguaje para la implementación del sistema, lo que nos ofrece también ciertas ventajas:

- Es un lenguaje independiente de la plataforma, es decir, podrá funcionar correctamente en ordenadores de todo tipo y con sistemas

operativos distintos (Windows, Linux, Mac, etcétera).

- Manejo automático de la memoria.
- Programación Orientada a Objetos. Los objetos agrupan en estructuras tanto los datos como los métodos que hacen uso de los mismos.
- Proporciona una colección de clases para su uso en aplicaciones de red, que permiten abrir sockets y establecer y aceptar conexiones con servidores o clientes remotos.
- Alta robustez, fue diseñado para la creación de software altamente fiable.

2.4.1. ECLIPSE

Eclipse, desarrollada originalmente por IBM y actualmente a cargo de la organización sin ánimo de lucro Eclipse Foundation, es una plataforma concebida para la integración de múltiples herramientas de desarrollo.

Figura 18: Logo Eclipse

Se trata de la plataforma escogida para la implementación de los diferentes módulos que formarán parte del sistema.

2.4.1.1. ¿Por qué Eclipse?

Esta decisión está íntimamente ligada al lenguaje utilizado, Java. Aunque la plataforma Eclipse no está limitada a un lenguaje específico, sí que tiene gran popularidad entre los desarrolladores que se dedican a Java.

2.4.2. JAVASCRIPT

JavaScript, publicado en junio de 1997, es un lenguaje de programación interpretado, dialecto del estándar ECMAScript. Se define como orientado a objetos, basado en prototipos, imperativo, débilmente tipado y dinámico. Se utiliza principalmente en su forma del lado del cliente (client-side), implementado como parte de un navegador web, lo que permite mejoras en la interfaz de usuario y páginas web. También se utiliza en otro tipo de aplicaciones como documentos PDF o widgets. Aunque su nombre invite a la confusión, Java y JavaScript tienen semánticas y propósitos diferentes.

Todos los navegadores modernos interpretan el código JavaScript integrado en las páginas web. Para interactuar con una página web se provee al lenguaje JavaScript de una implementación del Document Object Model (DOM).

Figura 19: Logo JavaScript

En este proyecto tendremos que extraer el código JavaScript incluido en las páginas webs para su posterior análisis, lo que dará como resultado la decisión de si está o no ofuscado.

2.4.3. HTML

HTML, de las siglas en inglés HyperText Markup Language, se trata de un

lenguaje de marcado para la elaboración de páginas web que se encuentra a cargo del World Wide Web Consortium (W3C). Actúa como estándar (establece la estructura básica y el código, también denominado HTML) para la descripción de los elementos contenidos en una página web tales como imágenes, vídeos, texto, etcétera. Está considerado como el lenguaje web más importante a nivel mundial y jugó un papel destacado en la aparición, desarrollo y expansión de la World Wide Web.

Figura 20: Logo HTML

Para la extracción del código JavaScript ejecutado por una página web, hemos de escanear primero todos los elementos de la misma, elementos de HTML

2.4.4. JUnit

JUnit, creado por Erich Gamma y Kent Beck, se trata de un conjunto de clases (framework) simple y de código abierto diseñado para la implementación y ejecución de clases Java de manera controlada. Esto permite evaluar el comportamiento de cada uno de los métodos de la clase.

Figura 21: Logo JUnit

En este proyecto se ha utilizado como entorno de pruebas en relación a la implementación de la herramienta con HTMLUnit.

2.4.5. OFUSCADORES

Se han utilizado algunos de los muchos ofuscadores disponibles en la web. En este caso se ha optado por cuatro herramientas que ofrecieran para un mismo código, resultados con ostensibles diferencias, ya que esto implica que se llevan a cabo diferentes técnicas de ofuscación y por tanto, los datos recogidos ofrecen una mejor representación. Otra de las condiciones es que fueran gratuitas:

- https://www.daftlogic.com/projects-online-javascript-obfuscator.htm
- http://www.danstools.com/javascript-obfuscate/index.php
- https://javascriptobfuscator.com/Javascript-Obfuscator.aspx
- http://javascript2img.com/

Figura 22: Logo DaftLogic, uno de los ofuscadores utilizados

Su papel ha estado limitado al entrenamiento. Con estas cuatro herramientas hemos podido ofuscar los ejemplos escogidos, lo que nos ha permitido extraer las características de un mismo código ofuscado y sin ofuscar para el posterior diseño del clasificador.

2.4.5.1. Diferencias y similitudes entre ellos

A grandes rasgos, todos persiguen los mismos objetivos: hacer que el código sea imposible de leer y dificultar que otros puedan copiar y pegar bloques de código, así como reducir el tamaño del archivo, lo que permite un ahorro de ancho de banda de salida para el propietario del sitio web del mismo modo que se reduce el uso de ancho de banda de entrada para los visitantes del mismo.

Para ello todos eliminan las líneas en blanco y compactan lo máximo posible el código, y cambian los nombres de variables y funciones por combinaciones de caracteres aleatorias.

Pero es interesante para el desarrollo del proyecto que también existan diferencias entre las herramientas elegidas. De esta forma https://javascriptobfuscator.com/Javascript-Obfuscator.aspx trabaja la codificación y el desplazamiento de strings y utiliza la codificación con hexadecimales en abundancia, http://www.danstools.com/javascript-obfuscate/index.php propone una ofuscación algo más avanzada con el uso de operaciones para representar constantes y la aplicación de una codificación propia, https://www.daftlogic.com/projects-online-javascript-obfuscator.htm muestra un resultado igual de efectivo pero más sencillo de obtener y http://javascript2img.com/obfuscating.php aumenta de forma importante el número de caracteres a pesar de eliminar los espacios en blanco, ya que introduce más términos aleatorios de gran longitud.

2.4.6. MATLAB

MATLAB (abreviatura de MATrix LABoratory), diseñada por Cleve Moler y lanzada inicialmente en 1984, es una herramienta de software matemático que ofrece un entorno de desarrollo integrado (IDE) con un lenguaje de programación propio (lenguaje M) basado en matrices. La plataforma de MATLAB está optimizada para resolver problemas de ingeniería y científicos.

Figura 23: Logo Matlab

En nuestro proyecto ha servido como programa de apoyo, realizar los cálculos más complejos y representar los datos que han servido para establecer los límites del clasificador.

2.5. PRESUPUESTO

Una vez se han definido la planificación, la metodología seguida y los requisitos tanto funcionales como no funcionales del sistema así como las tecnologías utilizadas, se procede a la realización de un estudio económico que arroje una cifra estimada del coste asociado al desarrollo íntegro del proyecto. Se diferenciarán cuatro apartados: material utilizado, software, personal implicado y los gastos indirectos.

2.5.1. MATERIAL UTILIZADO

A diferencia de otros, este proyecto no necesita de una gran inversión inicial para su puesta en marcha. No es necesario un laboratorio, ni material del mismo, tampoco placas, componentes electrónicos o cableados. El único material obligatorio del que se ha de disponer es un ordenador, en este caso se han utilizado:

- Portátil MSI GE60 2PE Apache Pro, se muestra en la ‘Figura 24: Equpo utilzado para la realización del proyecto’.

- Como apoyo también se ha hecho uso de un portátil Packard Bell EasyNote TM85.
- De manera indirecta, también ha sido necesario un router Comtrend CT-5365+.

Figura 24: Equpo utilzado para la realización del proyecto

Para la estimación de costes asociados al material debemos tener en cuenta varias cosas:

- El uso del ordenador de apoyo ha sido irrelevante por lo que descartamos la suma de ninguna cifra al presupuesto.
- El router es propiedad de la empresa de telecomunicaciones que suministra el servicio.
- El coste derivado del uso del ordenador portátil principal se puede estimar a partir de:
    - El precio inicial: 1.200 euros.

- Tiempo de vida: 18 meses.
- El tiempo que se ha utilizado para el desarrollo del proyecto: 5 meses.
- El porcentaje de tiempo empleado en otros menesteres: un 50% aproximadamente.

Con todo, tenemos que la cifra final se calcula como: 1.200 entre los 18 meses de vida, por los 5 de “desgaste” entre 2 porque el equipo no se ha destinado en exclusiva al proyecto: 167 euros.

2.5.2. SOFTWARE

Si nos centramos ahora en el plano del software, la gran mayoría de herramientas utilizadas en las distintas fases del proyecto son gratuitas (Eclipse, GanttProject, HTMLUnit, ofuscadores web). Pero hay una excepción, Matlab, aunque su versión para estudiantes incluye un importante descuento, que además se ve acentuado por la necesidad únicamente del programa base sin complementos.

En total 35 euros + IVA.

2.5.3. PERSONAL

Para obtener una cifra sobre el coste asociado al personal involucrado en la realización del proyecto tenemos que saber cuántas han sido las personas que han trabajado y qué cantidad de tiempo han dedicado, así como el precio de cada una de estas horas en función de su formación y cargo.

- Dos personas implicadas: el autor/desarrollador del proyecto, Lucas Cruz Cruz, y el supervisor/tutor, Gabriel Maciá Fernández.
- El proyecto ha durado cinco meses.
- Un salario acorde al mercado actual para un desarrollador recién titulado podría estar en torno a 1.000 euros, que por el tiempo

invertido nos deja un total de 5.000 euros.

- Durante estos cinco meses se han llevado a cabo reuniones periódicas con el supervisor cada quince días, un total de diez reuniones de aproximadamente una hora de duración.
- Los emolumentos correspondientes al tutor, doctor en la Universidad de Granada, se tasan en unos 100 €/h para un total de 1.000 euros.

2.5.4. GASTOS INDIRECTOS

La realización de un proyecto de tal magnitud lleva inherentemente asociados una serie de gastos indirectos. Estos hacen referencia al incremento en la factura de la luz, el mantenimiento del puesto de trabajo y los equipos utilizados, la conexión de acceso a Internet, seguro del ordenador principal, desplazamientos y otros.

Teniendo en cuenta que el autor del proyecto no ha residido en Granada durante estos meses, lo que implica una serie de viajes (150 Km solo ida), además de todo lo mencionado anteriormente, la cifra podría situarse en 1.200 euros.

2.5.5. COSTES TOTALES

Para finalizar se resumirán en la 'Tabla 2: Presupuesto' todos los costes explicados anteriormente. En total, para llevar a cabo el proyecto sería necesario un presupuesto de:

7.409,35 €, siete mil cuatrocientos nueve euros con treinta y cinco céntimos.

Concepto	Coste
Salario desarrollador	5.000 euros
Salario supervisor	1.000 euros
Material utilizado	167 euros
Software	42,35 euros
Gastos indirectos	1.200 euros
Total:	**7.409,35 euros**

Tabla 2: Presupuesto

3. DISEÑO E IMPLEMENTACIÓN

A continuación se expondrán los detalles del diseño e implementación del sistema desarrollado que será capaz en última estancia de detectar si un sitio web contiene código JavaScript ofuscado y por tanto, existe la posibilidad de que haya sido comprometido por un atacante.

3.1. ARQUITECTURA DEL SISTEMA

El sistema implementado se ha dividido en tres módulos, cada uno de los cuales lleva a cabo una tarea determinada. El primero estará encargado de extraer el código JavaScript que ejecuta la página que se quiere analizar, para ello deberá descargar la página y realizar una búsqueda de os elementos que lo contienen para finalmente presentar todo el código en un documento de texto. El segundo de los módulos se encargará de analizar este documento para extraer las características de código ofuscado. Cada característica será representada como un número, ocho en total, que se han de pasar al clasificador, que se incluye en el mismo módulo. Con este, y basándose en los datos del extractor, se tomará una decisión sobre si el código está o no ofuscado. Los resultados serán mostrados a través de una interfaz que facilitará a su vez la interacción del usuario con el sistema y que se establece como el tercer módulo. A grandes rasgos, el funcionamiento del sistema sería el siguiente (siguiendo el esquema de la ‘Figura 25: Funcionamiento del sistema’):

1) El usuario debe introducir la URL del sitio web que desea analizar y seleccionar en el desplegable qué navegador ha de emular el sistema (elegir entre el navegador por defecto, Google Chrome, Internet Explorer o Mozilla Firefox). Cuando esté listo pulsará el botón Aceptar.
2) En este momento, desde la interfaz se llama al único método de la clase *ExtraeJavascript, ObtenerJavascript*. Este método emulará el navegador elegido y descargará la página del sitio indicado, localizará los fragmentos de código JavaScript que contiene y los

presentará en forma de documento de texto.

3) Posteriormente, se ejecutará de nuevo desde la clase interfaz, el método principal de la clase *AnalizaJavascript*, nombrado como *ObtenerDatos.* Haciendo uso del resto de funciones de la clase se extraerán las características de código ofuscado que han sido implementadas, ocho en total, y pasará los resultados al clasificador que se ejecuta a continuación en el mismo método.

4) Los resultados tanto del extractor como la decisión del clasificador serán recibidos en la Interfaz que los mostrará cambiando su aspecto en función del resultado: pulgar arriba y color verde de fondo en caso de no detectar ofuscación del código y pulgar abajo y color rojo si se determina la ejecución de código JavaScript ofuscado.

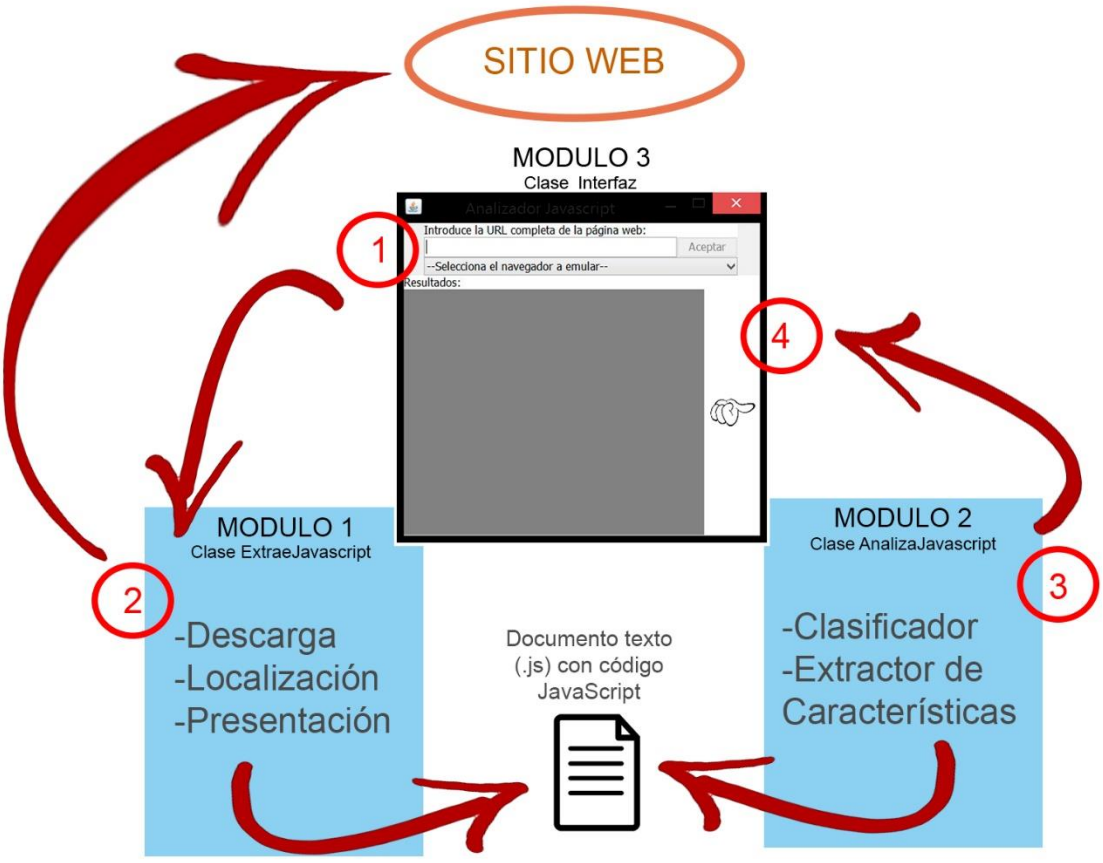

Figura 25: Funcionamiento del sistema

La siguiente 'Figura 26: Diagrama UML' muestra el diagrama UML, con sus clases y métodos correspondientes, del sistema que se explicará más en detalle en los puntos posteriores.

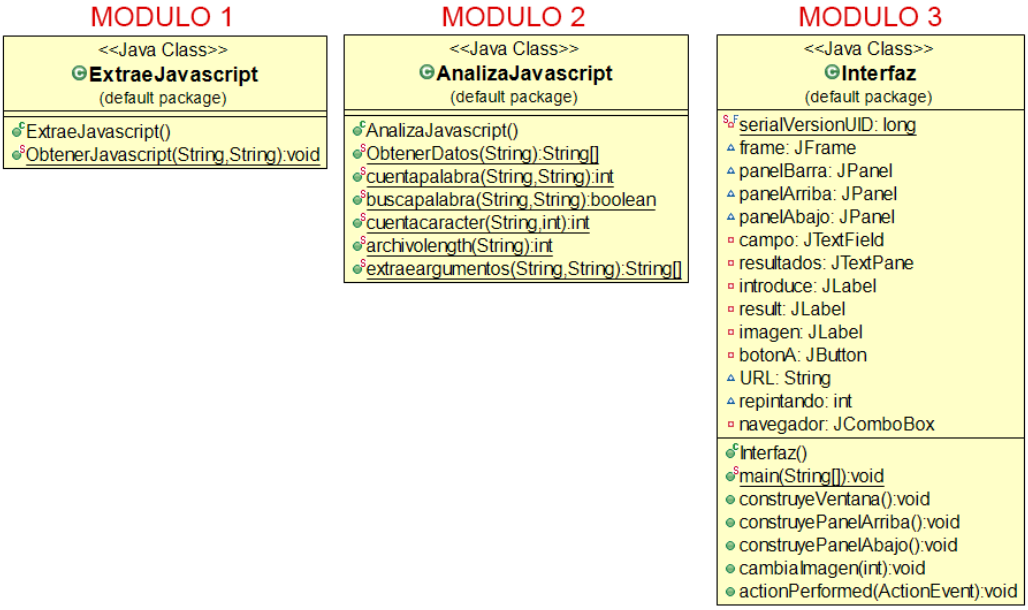

Figura 26: Diagrama UML

3.1.1. EXTRACCIÓN DE CÓDIGO JAVASCRIPT

El primer módulo del sistema consiste en un extractor de código JavaScript que presente en forma de documento de texto (*.js*) todo el código ejecutado por una página web al ser descargada.

3.1.1.1. Descarga

Lo primero que se debe hacer entonces es descargar la página con HTMLUnit, para lo cual se hace uso del objeto *WebClient* y el método *getPage(URL).* Si no se indica lo contrario al crear el objeto *WebClient*, HTMLUnit emulará un navegador por defecto, pero es posible utilizar varias versiones de Internet Explorer, Chrome y Firefox.

3.1.1.2. Localización

Una vez se ha comprobado que este sencillo método funciona correctamente, pasamos a extraer la totalidad código JavaScript de la página que se quiere analizar. Existen varios caminos que nos pueden llevar al mismo

destino, pero el más sencillo es crear una lista con todos los fragmentos de código presentes y después imprimirlos uno a continuación del otro. Antes de continuar es necesario introducir varios conceptos.

- DOM: de las siglas en inglés Document Object Model, se trata de una interfaz de plataforma que proporciona un conjunto estándar de objetos para representar documentos HTML, XHTML y XML. Un modelo sobre cómo pueden combinarse dichos objetos, y una interfaz para acceder a ellos y manipularlos.
- Etiqueta HTML: HTML utiliza etiquetas o marcas, que consisten en breves instrucciones de comienzo y final, mediante las cuales se determina la forma en la que deben aparecer los distintos elementos de la página (texto, imágenes, vídeos, etcétera).

HTMLUnit permite navegar entre los distintos elementos del DOM, identificándolos por su etiqueta. Para cumplir con el propósito de este módulo deberemos localizar todos aquellos marcados con la etiqueta “script” para lo cual haremos uso del método *getElementsByTagName*.

3.1.1.3. Presentación

Una vez se han localizado todos los elementos marcados como “script” incluidos en la página, hemos de tomar su contenido para simplemente imprimirlo con un buffer de escritura. Nos encontramos entonces con dos posibles situaciones:

- El código JavaScript está presente en el documento HTML. Solo debemos leerlo e imprimirlo en el documento donde presentaremos todos los fragmentos de código JavaScript.
- El código JavaScript no se encuentra en el documento. En este caso de ha de localizar el archivo con extensión *.js*, cuya dirección estará indicada en el campo *“src”*. Este puede apuntar a un directorio diferente dentro de la misma dirección de la página, por lo cual indicará una URL parcial o una dirección nueva. Para leer el contenido se precisará de una conexión a la nueva dirección y de buffer de lectura. A la par que se lee se puede ir escribiendo el código

en el documento de texto solución de esta primera parte.

3.1.1.4. Indicaciones

Es posible que trabajar directamente con páginas de Internet suponga un obstáculo difícil de salvar en un principio, especialmente si no se tiene muy claro el comportamiento de todos los métodos mencionados y las características de los elementos citados. Por lo que es recomendable la aplicación del módulo sobre de un pequeño documento HTML de diseño propio que nos permita realizar múltiples probaturas a fin de familiarizarnos con todos los agentes que participan en resolución del problema.

Por otro lado, notar que este primer módulo estará implementado como una clase, *ExtraeJavascript*, con un solo método, *ObtenerJavascript*, que es llamado cuando el usuario introduce una URL y pincha en el botón 'Aceptar' de la interfaz que se desarrollará como tercer y último módulo del sistema. Alternativamente, se podría implementar como un método dentro de la misma clase de la interfaz, aunque el funcionamiento no cambiaría en absoluto y la estructura planteada en tres módulos quedaría más difusa.

3.1.2. EXTRACCIÓN DE CARACTERÍSTICAS DEL CÓDIGO JAVASCRIPT

Esta parte estará dedicada al desarrollo del segundo módulo, aquel que analiza las características del código JavaScript recogido en el documento solución de la primera parte y con el que se pretende ofrecer una serie de datos que posibiliten la distinción entre código ofuscado y código no ofuscado. El segundo módulo está implementado sobre una clase llamada *AnalizaJavascript.*

3.1.2.1. Métodos básicos

Antes de comenzar con la implementación de las características en sí, es conveniente desarrollar algunos métodos básicos que facilitarán enormemente el trabajo a posteriori. En definitiva, lo que se trata es de hacer una valoración general sobre las características del código ofuscado a identificar y listar qué

métodos serían necesarios en cada caso, si alguno aparece varias veces será candidato a método básico según la terminología que se está utilizando.

Los métodos básicos implementados son:

- **Archivolength:** como bien se puede intuir por su identificador, este método devuelve el tamaño, medido en caracteres, del documento que contiene todo el código JavaScript que se ha de analizar. Su razón de ser está justificada por la necesaria normalización de los datos extraídos para cada característica, ya que en su mayoría dependen del número de apariciones de ciertas estructuras, y por tanto, es fundamental normalizar para poder comparar entre sí códigos de mayor y menor longitud.
- **Cuentapalabra:** este método mide el número de veces que aparece una palabra (o en su defecto, concatenación de caracteres) especificada, dentro del código JavaScript. Como se verá más adelante, es de gran utilidad para la extracción de varias características.
- **Buscapalabra:** en este caso y al contrario que en el método anterior, lo único que se obtendrá como respuesta es si la palabra especificada se encuentra o no en el documento.
- **Cuentacaracter:** su funcionamiento es similar a *cuentapalabra* pero esta vez obtendremos el número de veces que aparece un carácter determinado. Si bien no es especialmente útil en el caso de caracteres habituales como pueden ser las letras o los números, sí que puede ser interesante cuando se habla de caracteres especiales.
- **Extraeargumentos:** es posiblemente el más complejo de los métodos denominados como básicos. Su función consiste en extraer los argumentos de una función específica y devolverlos en forma de Array.

En el caso de implementar nuevas características, es posible que sea necesario ampliar la lista de métodos básicos, aunque con estos cinco debería

ser suficiente para el desarrollo de un módulo que satisfaga los requisitos planteados para el proyecto.

3.1.2.2. Implementación de características

Una vez se tienen los métodos básicos que se van a utilizar y se ha comprobado que cumplen su cometido correctamente (como siempre, un ejemplo controlado por el desarrollador es la mejor forma de hacerlo), se pasa a la implementación de las características. Esto se efectuará en el método principal de la clase, identificado como *ObtenerDatos,* requiere de un argumento de entrada, un string que indica la localización del documento de texto donde se incluyen todas las piezas de código JavaScript extraídas con el primer módulo.

Antes de seguir avanzando conviene explicar a qué se refiere la implementación de una característica. Como se ha explicado en anteriores apartados, el código ofuscado es el resultado de una transformación masiva y sistemática del código original. Por lo tanto es posible diferenciar un código ofuscado de uno que no lo está si atendemos a las modificaciones llevadas a cabo. Cuando se dice que se implementa una característica lo que se está diciendo realmente es que se implementa la búsqueda de algún elemento distintivo del código ofuscado. Como resultado lo que se obtiene es un dato que permite en algunos casos cuantificar en qué medida está presente dicho elemento distintivo, y en otros, conocer algún rasgo diferenciador del elemento.

Una vez que esto ha quedado claro, a continuación serán explicadas las características incluidas en el segundo módulo del sistema. Estas características están estrechamente relacionadas con las técnicas de ofuscación que han sido estudiadas, aunque en ocasiones es más efectivo acercar la mirada y fijarse en determinados detalles.

- **Ratio de definiciones y usos de strings:** se busca definir el ratio entre las invocaciones de funciones de JavaScript que se pueden utilizar para definir nuevas cadenas como *substring* y *fromCharCode* y el número de veces que se utilizan estos strings con funciones como *write* o *eval.* Con el objetivo de hacer el código mucho más lioso y dificultar la búsqueda de variables concretas, a menudo se

introducen múltiples definiciones de strings que no son utilizadas más adelante cuando se ofusca un JavaScript. A mayor ratio, aumentan las posibilidades de que el código esté ofuscado y viceversa. Para implementar esta característica se hace uso del método *cuentapalabra* y establecemos una relación entre las apariciones de *substring* y *fromCharCode* y las de *write* y *eval*. Una forma alternativa de calcular este ratio sería comprobar si las variables definidas con *substring* y *fromCharCode* aparecen posteriormente en el documento.

- **Número de ejecuciones de código dinámico:** se medirán tanto el número de llamadas a funciones que se utilizan para interpretar el código JavaScript de forma dinámica como *eval* y *setTimeout*, como el número de ejecuciones de funciones que pueden introducir cambios en el DOM tales como *document.write* y *document.createElement*. Funciones como las mencionadas aparecen con mayor frecuencia en los códigos ofuscados por ciberdelincuentes.

- **Tamaño del código evaluado dinámicamente:** es típico de los scripts maliciosos la utilización de la función *eval* para ejecutar de forma dinámica código con un alto índice de complejidad, en ocasiones con un tamaño de varios Kilobytes. Para implementar esta característica vuelve a ser necesario el uso de uno de los métodos básicos, *extraeargumentos*, el cual permite tener acceso integro a los argumentos que se han pasado a la función *eval.* Una vez se tiene el array con todos ellos, será tan sencillo como calcular la media del tamaño de los mismos, en caracteres.

- **Cantidad comentarios:** una de las técnicas de ofuscación enmarcadas dentro de la categoría de ofuscación aleatoria establece la introducción de comentarios como una de las modificaciones posibles. Aunque su impacto no es tan contundente como el de otras alternaciones del código JavaScript, se puede medir el número de comentarios que aparecen en el documento buscando en él estructuras como *// (doble barra)* y */* (barra*

asterisco) con el método *cuentapalabra* (al contar con dos caracteres, no es posible utilizar el método *cuentacaracter* tal y como se ha pensado).

- **Cantidad espacios en blanco:** dentro de la misma categoría de ofuscación aleatoria se incluye otra técnica por la cual se introducen espacios en blanco para dificultar la comprensión del código. En esta ocasión se hará uso del método *cuentacaracter* para localizar el número de caracteres correspondientes al espacio en blanco presentes. También se aprovechará el método *cuentapalabra* para localizar diferentes formas de espacio en blanco.
    - \t: tabulación.
    - \n: nueva línea.
    - \r: retroceso de carro.
- **Divisiones de String:** pasamos ahora a la categoría de ofuscación de datos. Una de las técnicas hace referencia a la inclusión de divisiones de strings, lo que aumenta la complejidad del código ofuscado. Para lo cual implementaremos la búsqueda de estructuras como *"+"* (comillas, signo suma, comillas) y *""""* (correspondiente a un string vacío) nuevamente aprovechando el método básico *buscapalabra.*
- **Cantidad de operaciones:** siguiendo con la ofuscación de datos, como se ha explicad los números suelen expresarse en forma de operación matemática para dificultar los cálculos manuales y complicar aún más la lectura del código una vez ofuscado. Bastará con contabilizar, gracias al método *cuentacaracter*, el número de veces que aparecen los símbolos *+, -,* * y / para obtener una cifra aproximada de la cantidad de operaciones que se realizan a lo largo de todo el documento.
- **Cantidad de hexadecimales y Unicode:** pasando a técnicas pertenecientes a la ofuscación por codificación, una de las más sencillas de aplicar a un código si se quiere lograr una ofuscación

efectiva es el cambio de codificación que sustituye caracteres habituales por caracteres expresados en hexadecimal y/o Unicode. La implementación de esta característica se centra en contabilizar las veces que se encuentran las estructuras *0x*, *x* y *u,* ya que estas siempre aparecen antes de un carácter hexadecimal o Unicode.

3.1.2.3. Apreciaciones

Como se ha explicado anteriormente en este mismo apartado, cada característica implementada nos ofrece como resultado un dato. Para dar por finalizado este módulo del sistema debemos devolver todos los datos de forma conjunta, en este caso en un string que se mostrará en la interfaz que diseñaremos más adelante (no es estrictamente necesario pero supone una aportación adicional para el usuario más allá de la decisión sobre la ofuscación o no del código). No será necesario devolver los datos en sí ya que su uso solo será precisado más adelante por el clasificador, y este se implementará como parte del mismo método, *ObtenerDatos*, de la clase *AnalizaJavascript*.

Para un funcionamiento óptimo, se ha de comprobar en detalle el comportamiento de cada característica por separado. Con tal propósito se analizarán varias webs cuyo código JavaScript sea fácilmente accesible (hay multitud de ejemplos disponibles por Internet), lo que permitirá la verificación en primera persona de los resultados obtenidos por el sistema. El modo desarrollador del navegador Mozilla Firefox puede ser de gran ayuda, ya que muestra el documento HTML de las webs descargadas. Otra opción sería la de ejecutar por separado el primer módulo implementado, y comparar los resultados aportados por el segundo módulo con lo observado en el documento de texto que contiene el código JavaScript extraído (si el editor de texto utilizado cuenta con una herramienta de búsqueda, se podrán examinar las coincidencias existentes sin la necesidad de repasar todo el documento manualmente).

Un último apunte importante, los datos obtenidos de la implementación de cada característica han de ser normalizados, en caso de que el resultado dependa del tamaño del archivo, antes de imprimirse o ser utilizados por el clasificador. Para ello ha sido creado el método *Archivolength.*

3.1.3. CLASIFICADOR

El primero de los objetivos de este proyecto es desarrollar un módulo de clasificación de código JavaScript que permita la detección de código ofuscado. Hasta ahora se ha logrado la extracción de dicho código, también se ha diseñado un analizador que determina sus características con tan solo ocho datos numéricos y se ha probado sobre un número considerable de muestras para ver como difieren los resultados del análisis sobre un código sin ofuscar y el mismo ofuscado con varias herramientas de ofuscación. Con el clasificador se ha de recoger toda esta información y tomar la decisión final: SI está ofuscado o NO está ofuscado, o lo que es lo mismo, SI puede estar siendo atacado o por el contrario NO hay indicios de que pueda ser víctima de un ataque.

3.1.3.1. Clasificador Naive Bayes

Naive Bayes es un algoritmo de clasificación basado en el teorema de Bayes que considera que: todas las características consideradas para la clasificación son independientes del valor de cualquier otra característica. Sin embargo, las características no siempre son independientes, un defecto que le vale al clasificador bayesiano el apelativo de “ingenuo” (Naive en inglés).

Comenzando por el principio, el teorema de Bayes, que toma su nombre del filósofo inglés Thomas Bayes, establece la probabilidad condicional de un evento aleatorio V dado A en términos de la distribución de probabilidad condicional del evento A dado V (que puede tomar n valores V_i) y la distribución de probabilidad marginal de sólo V. En otras palabras, vincula la probabilidad de V dado A con la probabilidad de A dado V según la siguiente ecuación:

$$P(V|A) = \frac{P(A|V)P(V)}{P(A)}$$

Donde:

- $P(V) \rightarrow probabilidades\ a\ priori$
- $P(A|V) \rightarrow probabilidad\ de\ A\ en\ la\ hipótesis\ V$
- $P(V|A) \rightarrow probabilidades\ a\ posteriori$

Partiendo de esta base, el clasificador Naive Bayes selecciona la clasificación de mayor probabilidad de V_{nb} dados los valores de los atributos a_1, a_2,…, a_n.

$$V_{nb} = argmax_{v_j \in V} P(v_j) \prod P(a_i|v_j) \quad (1)$$

Se estima $P(a_i|v_j)$ como:

$$P(a_i|v_j) = \frac{n_c + mp}{n + m} \quad (2)$$

Donde:

- $n \rightarrow número\ de\ ejemplos\ de\ entrenamiento\ para\ los\ que\ v = v_j$
- $n_c \rightarrow número\ de\ ejemplos\ para\ los\ que\ v = v_j\ y\ a = a_i$
- $p \rightarrow probabilidad\ a\ priori\ para\ P(a_i|v_j)$
- $m \rightarrow tamaño\ equivalente\ de\ la\ muestra$

3.1.3.1.1 ¿Por qué Naive Bayes y no otro clasificador?

La sección de estadística y probabilidad juega un papel clave en el sistema que se está desarrollando, pero no es considerado como uno de los protagonistas del mismo, por lo cual, se optará por una solución básica, dejando abierta la posibilidad de experimentar con clasificadores de mayor complejidad que pudieran mejorar la precisión de los resultados obtenidos para un futuro proyecto.

La elección del clasificador bayesiano y no cualquier otro de los muchos que existen atiende principalmente a una razón: su sencillez. Además, hay algunas características más del clasificador Naive Bayes que encajan a la perfección con el comportamiento y la naturaleza del sistema implementado:

- Es fácil de entrenar: incluso con un número relativamente pequeño de muestras como el que se ha usado, los resultados pueden ser óptimos.
- Es rápido: ideal para el sistema desarrollado, ya que es necesario ofrecer una respuesta casi instantánea al usuario.

- Es insensible a características irrelevantes: incluir una característica irrelevante no afecta negativamente al comportamiento del clasificador, esto ayuda si tenemos en cuenta el número de características implementadas, cada una con un impacto determinado no conocido.

3.1.3.2. Entrenamiento

Una vez obtenido el documento con todo el código JavaScript de la página web descargada y después de calcular los resultados del análisis de todas las características implementadas, llega la fase de entrenamiento, donde se pondrán a prueba los dos módulos desarrollados hasta el momento con la finalidad de reunir los datos necesarios para la aplicación posterior del clasificador probabilístico Naive Bayes que se encargará de darles sentido.

3.1.3.2.1 Elección de muestras y herramientas de ofuscación

Una de las claves del entrenamiento que se va a ejecutar a continuación está en la elección de las muestras (sitios web) y las herramientas de ofuscación que se usarán. Cuantas más muestras y herramientas diferentes participen en esta fase más precisos serán los datos que se obtendrán.

Se recomienda, como mínimo, utilizar cinco muestras de extensión y orígenes diferentes y al menos tres herramientas de ofuscación. El abanico para elegir es muy amplio por lo que se han de tener en cuenta algunas cosas:

- Que los ejemplos funcionen perfectamente en nuestro sistema: como explicaremos en el apartado de “vías futuras”, el navegador programado con HTMLUnit tiene algunas limitaciones, por lo que debemos comprobar que nuestro sistema se comporta correctamente cuando introducimos la dirección deseada antes de escoger dicha web como muestra.
- Muestras diferentes: lo ideal es que se incluyan muestras con características muy diferentes entre sí para cubrir un mayor rango de acción. Por ejemplo se podría usar un mix de webs como el buscador de Google, más compleja, y alguna que contenga ejemplos

de códigos JavaScript más sencillos.

- Herramientas de ofuscación gratuitas: existen multitud de herramientas de ofuscación en Internet, muchas de ellas de pago. Estas suelen ofrecer mayores posibilidades de configuración, pero no es algo estrictamente necesario para la realización del proyecto que se está desarrollando.
- Herramientas de ofuscación sencillas y con comportamientos distintos: de entre todas las herramientas gratuitas disponibles, que no son pocas, se han de escoger aquellas que sean más fáciles de usar y a su vez que apliquen técnicas diferentes de ofuscación. En definitiva, si tenemos cuatro herramientas y dos de ellas nos aportan una solución similar, descartaremos aquella que sea más engorrosa de utilizar y buscaremos una alternativa.

En el caso que nos ocupa hemos seleccionado ocho muestras, cinco de la web https://www.uv.es/jac/guia/jscript/javascr.htm, que contiene variedad de ejemplos JavaScript, sin ofuscación ninguna. Los otros tres corresponden con webs conocidas y de mayor entidad como https://www.google.es, https://es.wikipedia.org/, y https://www.youtube.com/. Las herramientas de ofuscación elegidas son las indicadas en el punto 4.7:

- https://www.daftlogic.com/projects-online-javascript-obfuscator.htm
- http://www.danstools.com/javascript-obfuscate/index.php
- https://javascriptobfuscator.com/Javascript-Obfuscator.aspx
- http://javascript2img.com/

3.1.3.2.2 Procedimiento

El objetivo de este enteramiento es, como se ha explicado con anterioridad, el de obtener los datos necesarios para la configuración de un clasificador probabilístico. Estos datos serán el resultado de analizar con el sistema implementado las ocho muestras mencionadas tanto en su forma original como después de pasar por cada uno de los ofuscadores que forman parte de nuestra selección. Lo que dará lugar a un total de 40 resultados, 8 sin ofuscar y 32

ofuscados (8 por cada uno de las 4 herramientas de ofuscación) para cada una de las características.

Los pasos a seguir para que el entrenamiento resulte satisfactorio y a su vez, esté lo suficientemente automatizado para que introducir nuevas muestras o cambiar de ofuscador en el futuro no resulte una tarea tan laboriosa como lo será en un principio, se detallan en los siguientes párrafos. Primero se ha de:

1) Aplicar el primer módulo del sistema a la primera de las webs seleccionadas de modo que obtengamos el documento de texto con su código JavaScript correspondiente.

2) Copiar este código JavaScript en cada uno de los ofuscadores y ejecutar la herramienta, obtendremos cuatro códigos ofuscados que se han de copiar, cada uno por separado, en un documento de texto con extensión *.js*.

3) Repetir los puntos 1 y 2 para las siete muestras restantes.

Una vez se tienen los 40 códigos JavaScript, preferiblemente en una misma carpeta, toca modificar (en una clase aparte que servirá de apoyo únicamente) el segundo de los módulos que se habían implementado, de forma que no sea necesario ejecutarlo manualmente para obtener los resultados correspondientes a cada uno de ellos. La clase creada se llamará *AnalizaJavascript2* y los cambios a introducir son:

4) A partir de *AnalizaJavascript* creamos la clase *AnalizaJavascript2,* que funcionará de manera autónoma, por lo que el método *ObtenerDatos* pasa a ser *main.* Tampoco es necesario ahora pasar ningún argumento ni devolver variable alguna con el *return*.

5) Definimos un array donde cada posición es un string que apunta a uno de los 40 documentos generados en los pasos anteriores.

6) Dónde antes se ejecutaba una sola vez el analizador con el que obteníamos los resultados para cada característica implementada, ahora será necesario un bucle que analice los 40 documentos.

7) Por último, se creará un documento (con un buffer de escritura) donde se recogerán en forma de matriz los resultados de todas las

características para cada uno de los 40 códigos. Lo más importante es que este documento tenga un formato que sea fácil de interpretar para Matlab, programa con el que representaremos la información obtenida.

3.1.3.2.3 Análisis del entrenamiento

Si todo va bien, después de ejecutar el analizador auxiliar (*AnalizaJavascript2*) tendremos un documento con todos los datos dispuestos en una matriz de 40x8 correspondientes al número de códigos analizados (filas) y el número de características implementadas (columnas). Seguidamente se ha de introducir esta matriz como una variable en Matlab, algo muy sencillo gracias a la función *load* con la que ya cuenta el software de carácter profesional.

Después, se tendrán que realizar algunas operaciones para separar los datos en función de la característica a la que hacen alusión ya que serán representadas de forma individual. Cuando se dibujen las gráficas es importante que se puedan distinguir de alguna manera qué puntos representan un dato que corresponde a un código ofuscado y cuáles son resultado del análisis de un código sin ofuscar. Un cambio en la forma y/o el color debería ser más que suficiente.

Las gráficas obtenidas se mostrarán para finalizar este apartado, pero antes se ha de tener en consideración que el eje X carece de relevancia ya que únicamente refleja el número de la muestra representada. Aun así, se ha optado por no representar los datos en una única dimensión por comodidad a la hora de identificar a simple vista los puntos que corresponden con muestras ofuscadas y no ofuscadas. Por esta misma razón se puede apreciar que:

- Las cruces azules (+), que representan los resultados del análisis de los códigos ofuscados (recordar que son ocho muestras por cuatro herramientas diferentes que han sido utilizadas) alcanzan el índice 32.
- Los asteriscos rojos (*), que representan los resultados del análisis de los códigos no ofuscados (muestras sin ofuscar con ninguna de las herramientas), se quedan en el índice 8.

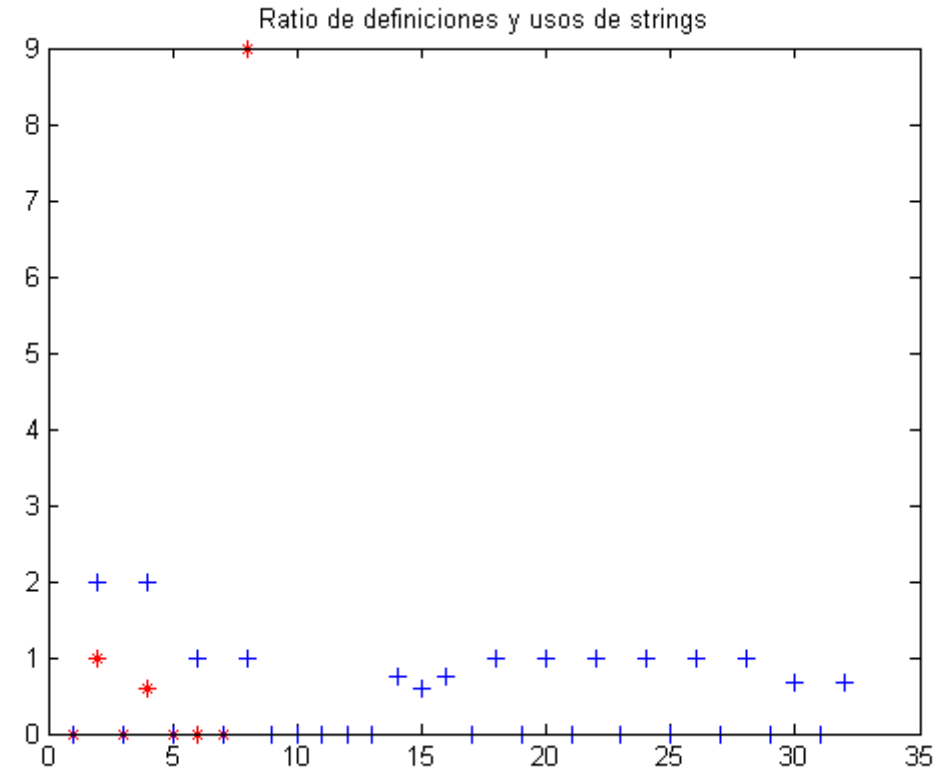

Figura 27: Ratio de definiciones y usos de strings – entrenamiento

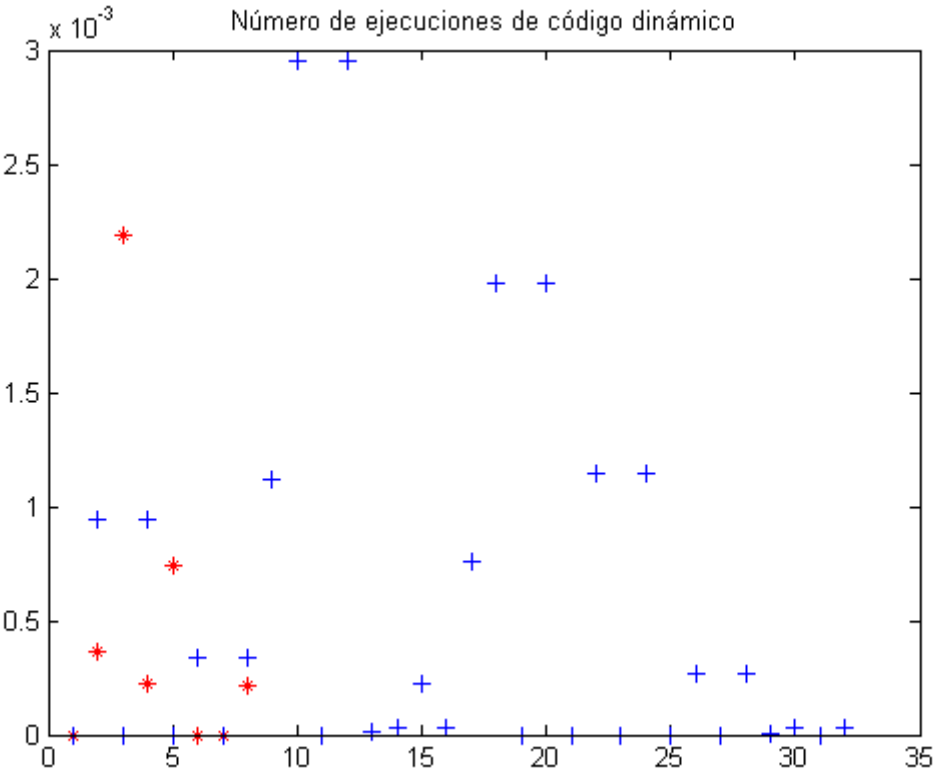

Figura 28: Número de ejecuciones de código dinámico - entrenamiento

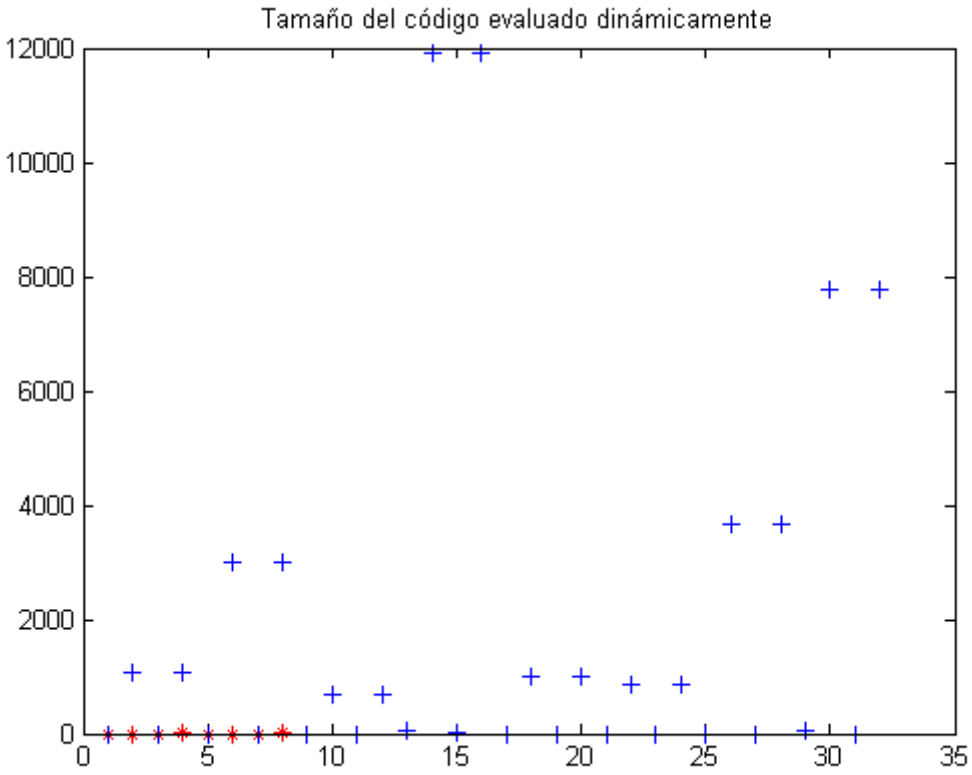

Figura 29: Tamaño del código evaluado dinámicamente - entrenamiento

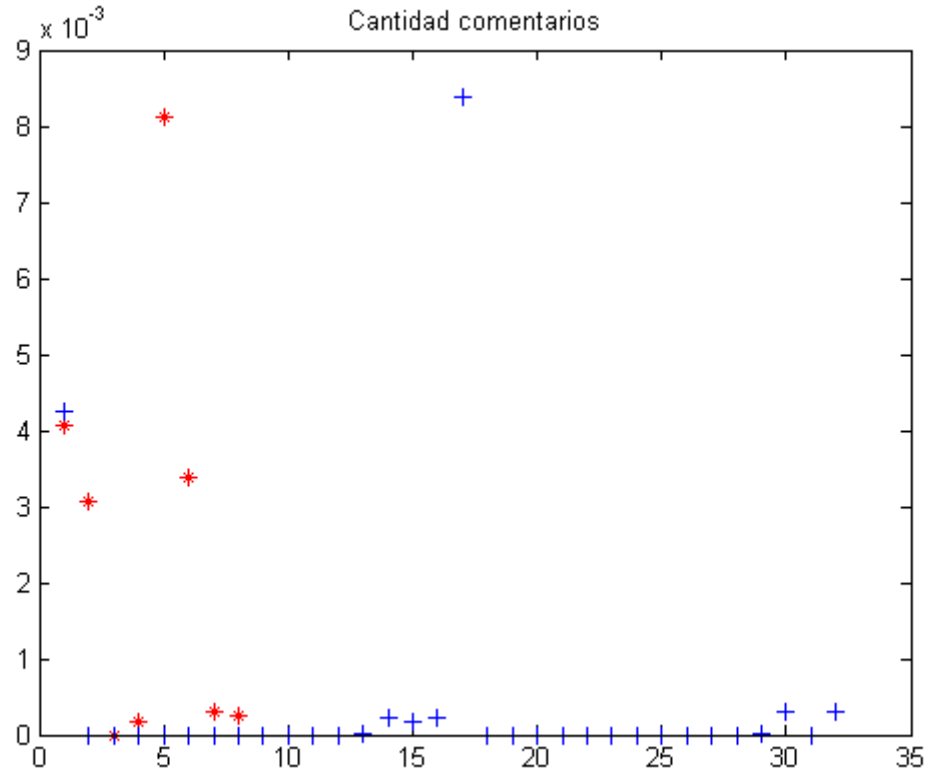

Figura 30: Cantidad de comentarios - entrenamiento

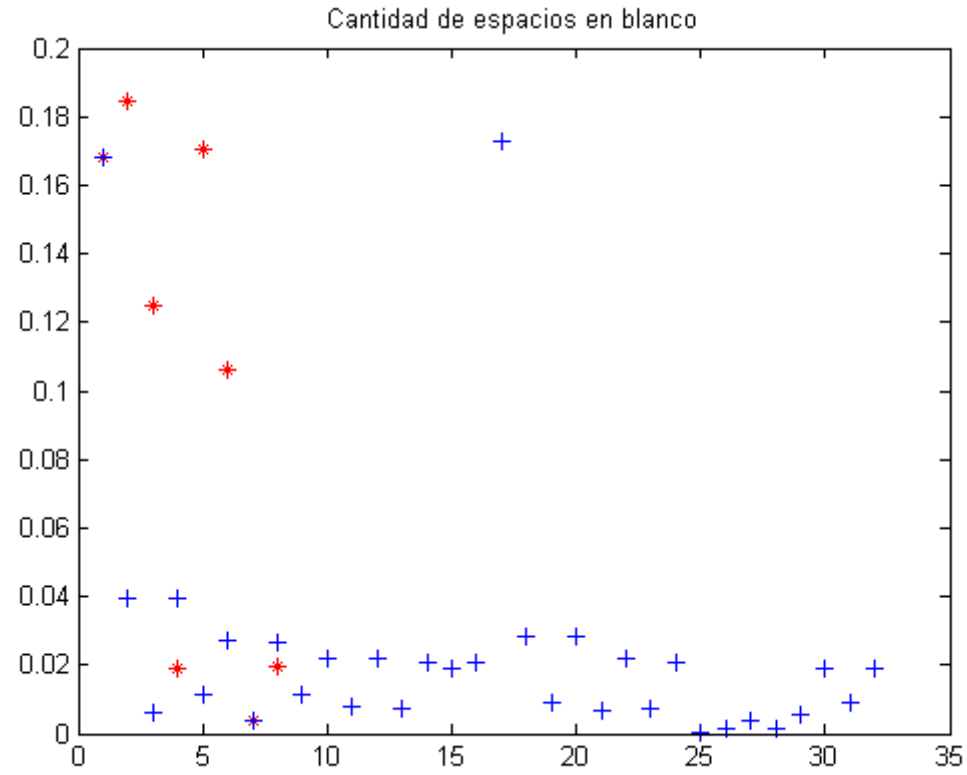

Figura 31: Cantidad de espacios en blanco - entrenamiento

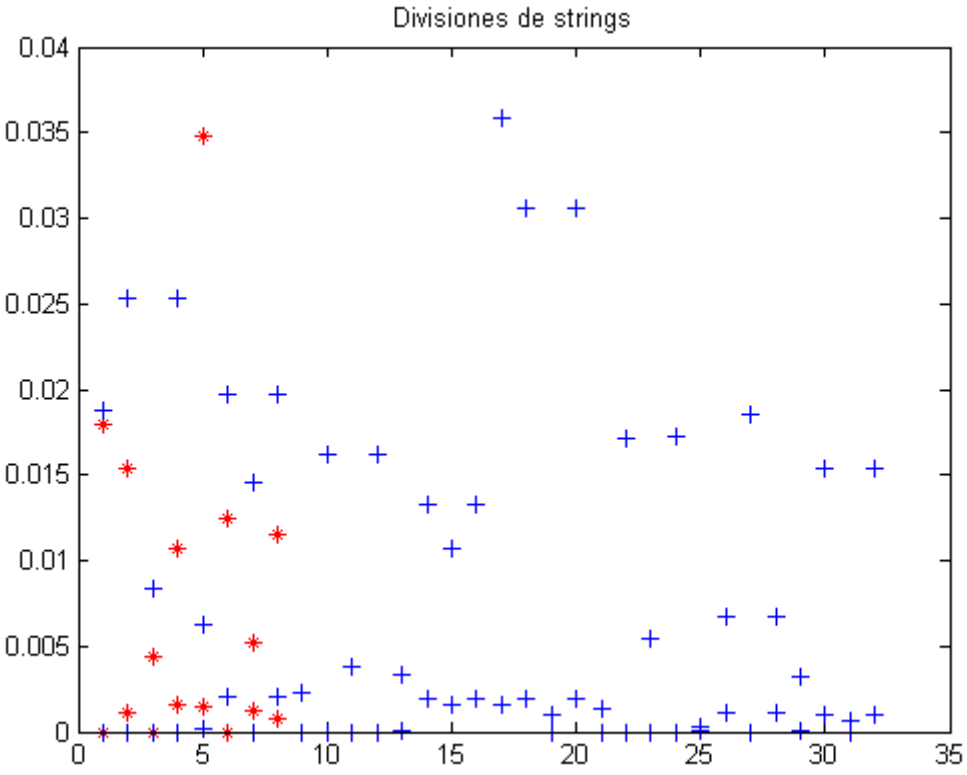

Figura 32: Divisiones de strings - entrenamiento

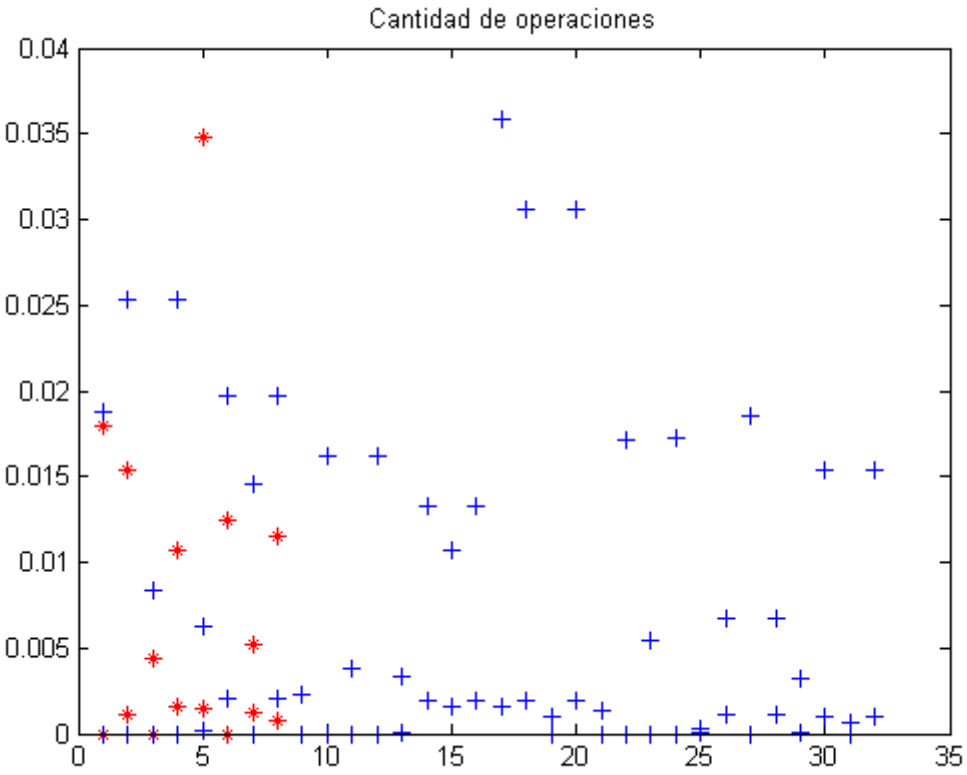

Figura 33: Cantidad de operaciones - entrenamiento

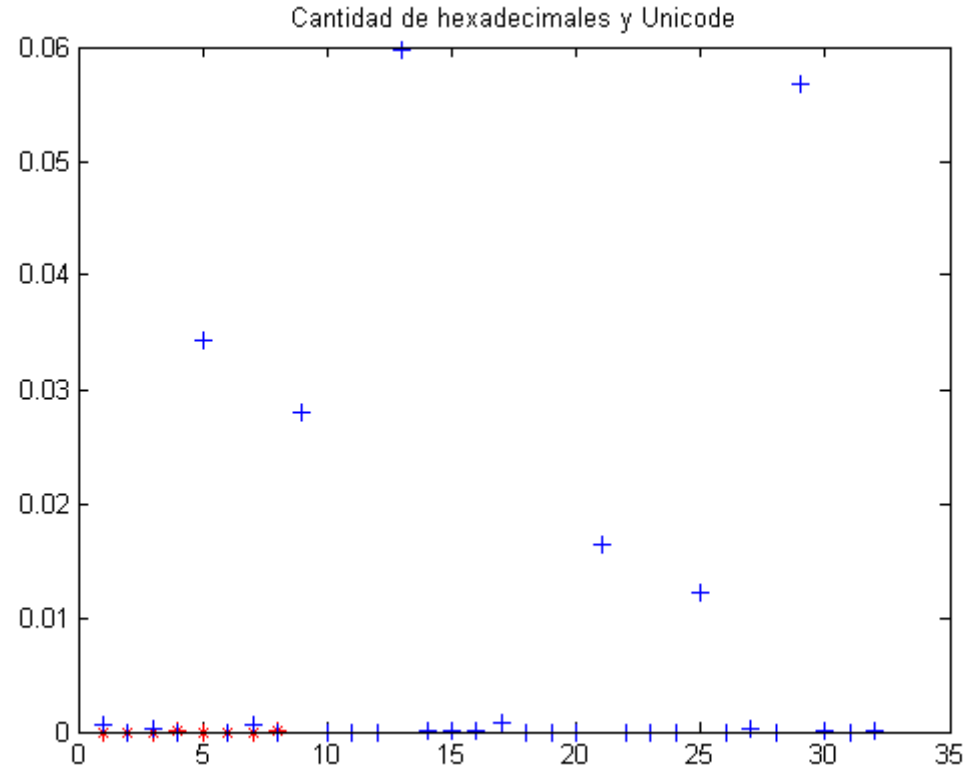

Figura 34: Cantidad de hexadecimales y Unicode – entrenamiento

Se puede apreciar como algunas de las características, véase por ejemplo el tamaño del código evaluado dinámicamente o la cantidad de hexadecimales y Unicode, muestran al ser representadas una clara diferencia entre el comportamiento de los códigos ofuscados y los no ofuscados. Serán estas las que más peso obtengan en la decisión final. Por el contrario, hay algunas como la cantidad de operaciones y las divisiones de string que no reflejan un cambio abrupto, aun así tendrán su parte de culpa en la decisión aunque su participación tendrá un menor impacto.

Es una de las razones por las que se ha escogido Naive Bayes, ya que este clasificador toma cada característica por separado, partiendo de la base de que todas son válidas. Sin embargo, si una característica no revela diferencias entre el código ofuscado y el no ofuscado, simplemente no tendrá casi efecto en la decisión (la probabilidad final se multiplicará por un número similar) mientras que aquellas donde la diferencia es evidente tendrán un gran efecto (se multiplicará por número más distantes). En definitiva, todas suman, ninguna característica afecta negativamente al resultado, esto será clave para la aplicación del clasificador al sistema, también para el trabajo futuro que pueda efectuarse, ya que pueden incorporar infinitas características sin que el comportamiento del clasificador se vea deteriorado en ningún caso, existiendo a su vez posibilidad de mejorarlo si se da con una característica acertada.

3.1.3.3. Aplicación del clasificador al sistema

En este punto se especificará cómo el clasificador bayesiano que se ha expuesto de forma teórica en apartados anteriores se aplica al sistema desarrollado en este proyecto una vez se tienen los datos del entrenamiento.

Lo primero es establecer las características que serán tenidas en cuenta por el clasificador bayesiano. Partimos de las gráficas obtenidas en el entrenamiento. Como se ha explicado anteriormente, no entra dentro de los objetivos del proyecto el diseño de un clasificador demasiado complejo, bastará por tanto con marcar una línea en cada gráfica (son 8 en total) y agrupar los puntos en dos grupos: los que superan el corte y los que se quedan por debajo. En términos del clasificador, esto supone establecer dos características por cada gráfica (correspondientes a las ocho características de código ofuscado

implementadas).

¿Dónde situar el corte?

En una situación ideal, el entrenamiento habría dado como resultado ocho gráficas, y en cada una se podrían apreciar dos grupos de puntos claramente diferenciados, uno con los puntos que representan a las muestras de código ofuscado y otro con aquellos que representan las muestras de código no ofuscado. En esta situación teórica, el corte estaría situado justo en la mitad de ambos grupos, pero este proyecto no trata sobre un ejercicio teórico sino sobre un caso real, y por tanto, los grupos están dispersos. Entonces, la respuesta a la pregunta sería: el corte debe situarse a una altura en la cual cada grupo quede lo más unido posible y a su vez, la distancia entre ambos grupos sea la mayor posible.

Puede parecer complicado pero en la práctica se trata de intentar que la mayoría de puntos rojos estén por encima de la línea y la mayoría de los azules por abajo, o al revés. Como este proceso se ha realizado de forma manual, se recomienda una observación minuciosa de las gráficas.

Con todo, las características que tendrá en cuenta el clasificador bayesiano quedan tal y como se muestra en la siguiente tabla:

	Ratio >0	**NODCE >0,0005**	**LDEC >0**	**CC <0,001**	**CEB <0,04**	**CO >0,015**	**DS >0,0015**	**CHU >0**	**Tot**
SI	16	10	20	30	30	16	6	16	32
NO	3	2	1	4	3	2	1	1	8

Tabla 3: características calasificador bayesiano

Donde cada columna hace referencia a una característica implementada y el corte/condición establecido (resultados normalizados según el tamaño del documento) y las filas SI (ofuscado) y NO (ofuscado) se refieren al número de muestras ofuscadas y no ofuscadas que cumplen dicha condición. Notar que:

- Ratio: ratio entre definiciones y usos de strings

- NODCE (Number Of Dinamic Code Executios): número de ejecuciones de código dinámico.
- LDED (Length of Dynamically Evaluated Code): tamaño del código evaluado dinámicamente.
- CC: Cantidad de Comentarios.
- CEB: Cantidad de Espacios en Blanco.
- CO: Cantidad de Operaciones.
- DS: Divisiones de Strings.
- CHU: Cantidad de Hexadecimales y Unicode.
- Tot: muestras totales utilizadas.

A partir de la tabla anterior obtenemos los datos probabilísticos:

	Ratio >0	**NODCE >0,0005**	**LDEC >0**	**CC <0,001**	**CEB <0,04**	**CO >0,015**	**DS >0,0015**	**CHU >0**	**Tot**
SI	0,5	0,3125	0,625	0,9375	0,937	0,5	0,1875	0,5	0,8
NO	0,375	0,25	0,125	0,5	0,375	0,25	0,125	0,12	0,2

Tabla 4: Características clasificador bayesiano - probabilidades

Cosas a tener en cuenta antes de seguir:

- Establecer un corte como hemos hecho genera en realidad dos características a evaluar por el clasificador, pero una es complementaria de la otra, y por eso se han omitido de la tabla. Por ejemplo, tenemos la característica Ratio > 0 con probabilidad 0,5 para códigos ofuscados y 0,375 para no ofuscados, por consiguiente existe otra que sería Ratio <= 0 con probabilidad 0,5 para códigos ofuscados y 0,625 para los no ofuscados. A pesar de que no aparecen en la tabla, se tienen igualmente en cuenta.
- Las probabilidades de la columna “Tot” corresponden a P(Ofuscado)

y P(NoOfuscado). O lo que es lo mismo, probabilidad de que un código sea ofuscado y probabilidad de que un código no sea ofuscado. Aplicado sobre las muestras de nuestro entrenamiento estas probabilidades serían, como se refleja en la tabla, de 0,8 y 0,2. Pero el clasificador está pensado para actuar sobre cualquier código que se introduzca en el sistema, por lo tanto, estas dos probabilidades serán obviadas, lo que equivale a utilizar una probabilidad a priori de 0,5 y 0,5, esto significa suponer que un código tiene las mismas posibilidades de estar ofuscado que de no estarlo.

- Al contrario de lo que ocurre con el resto, los resultados de la característica CEB (Cantidad de Espacios en Blanco) no se corresponden con los esperados. A priori, la técnica de ofuscación de la que se deriva esta característica establece la introducción de espacios en blanco aleatorios, pero después de realizar el entrenamiento se observa que los códigos ofuscados tienen, en general, un índice CEB mucho más bajo que las muestras no ofuscadas. Esto se debe a que las herramientas utilizadas para ofuscar el código están pensadas para su uso por parte de 'web masters' que desean añadir un poco de seguridad a sus páginas. Además de reforzar la seguridad del sitio, estas herramientas suelen contar con características adicionales que ayuden a los responsables del mismo, una de las más habituales trata de compactar el código con el objetivo de reducir su tamaño y por tanto, agilizar la descarga de la página web en cuestión. No solo no añade nuevos espacios en blanco, sino que los elimina. No podemos comprobar que el *modus operandi* de los ciberdelincuentes sea diferente por lo que se adecuará el comportamiento del sistema según los datos del entrenamiento a pesar de que estos puedan ser contradictorios con el apartado teórico.

3.1.3.4. Implementación

Para terminar se ha de introducir el clasificador diseñado en el sistema a

fin de que este pueda tomar una decisión, condición indispensable para que el proyecto finalice satisfactoriamente. Como se ha comentado anteriormente, el clasificador bayesiano estará implementado dentro del segundo módulo, método *ObtenerDatos* de la clase *AnalizaJavasript.* El funcionamiento del clasificador sería el siguiente:

1) Después de descargar una página y extraer el código JavaScript, una vez se ha analizado y se han calculado los resultados correspondientes a cada característica implementada, se pone en funcionamiento el clasificador.
2) Los resultados del análisis se almacenan en las variables correspondientes y como el clasificador se incluye en la misma clase, tendrá acceso directo a todas ellas.
3) Como la probabilidad de que el código sea ofuscado es la misma de que sea no ofuscado, ambas parten en $P(v_j) = 0{,}5$.
4) Se toma el resultado obtenido de analizar la primera característica y se comprueba la condición correspondiente establecida en el apartado anterior.
    a. Si la cumple, la probabilidad de ofuscado y la de no ofuscado multiplica por la probabilidad correspondiente de la *"Tabla 4*: Características clasificador bayesiano - probabilidades*".* Por ejemplo, se ha obtenido un ratio de 0'1, cumple la condición pues es mayor que 0, por lo tanto la probabilidad de que esté ofuscado será 0'5*0'5 mientras que la probabilidad de que no esté ofuscado quedará como 0'5*0'375.
    b. Si no se cumple, ambas se multiplican por 1-p, siendo p en este caso la probabilidad de que se cumpla. Siguiendo el ejemplo, analizamos otro código y ahora se obtiene como ratio 0, no se cumple, por lo tanto la probabilidad de que esté ofuscado será 0'5*0'5 mientras que la probabilidad de que no esté ofuscado quedará como 0'5*0'625.
5) Se repite el punto 4 para las siete características restantes.

6) Al final, se comparan la probabilidad de que esté ofuscado con la probabilidad de que no lo esté y se escoge la que sea mayor.

7) Se devuelve la decisión.

3.2. INTERFAZ

El sistema ya funciona, pero aún falta diseñar una interfaz que permita a cualquier usuario, independientemente de sus conocimientos, su interacción con el mismo.

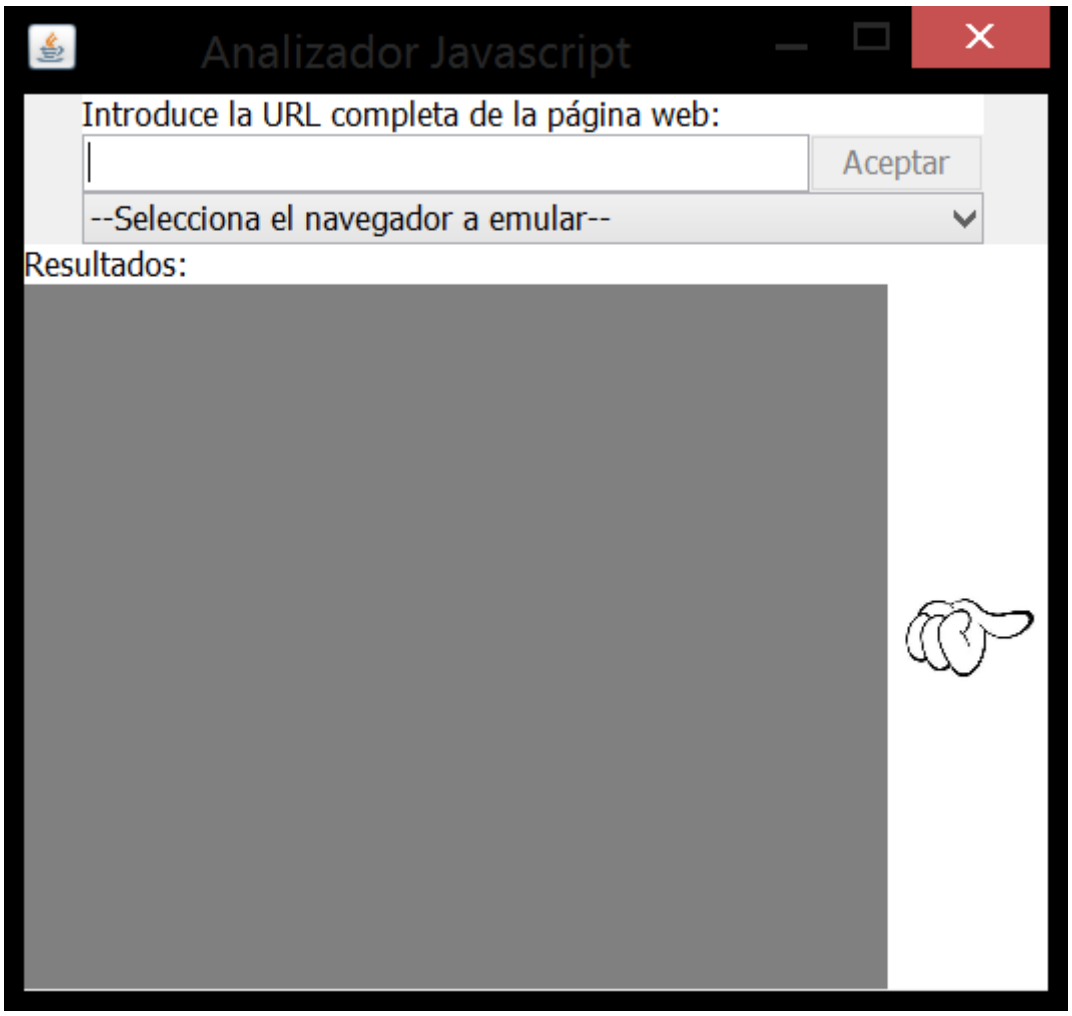

Figura 35: Interfaz

La interfaz propuesta (‘Figura 35: Interfaz’) no solo busca que el sistema sea accesible para cualquiera que tenga interés en comprobar si una web ejecuta

de alguna manera código JavaScript ofuscado, sino que además, se ha tenido muy en cuenta un aspecto que resulta fundamental en todas las aplicaciones que usamos a diario actualmente, el aspecto visual.

Aunque no se ha descartado su uso por parte de usuarios con mayores conocimientos en la materia, de ahí que por ejemplo, se impriman los resultados del análisis, sí que se cree particularmente interesante que la decisión final no solo se muestre en forma de texto sino que también se deje constancia de una forma más directa y clara.

3.2.1. MANUAL DE USO

A continuación se expondrá una breve descripción de los diferentes elementos que componen la interfaz y se explicarán las posibilidades de interacción del usuario con cada uno de ellos a modo de manual de uso. La interfaz cuenta con:

- Un cuadro de texto con la etiqueta “Introduce la URL completa de la página web:”: Como se puede intuir por la descripción, será en este cuadro de texto donde el usuario escribirá/pegará la dirección del sitio web que se desea analizar.
- Un botón con la etiqueta “Aceptar”: su cometido es simple, una vez se ha introducido la dirección del sitio web, el usuario deberá pulsar este botón para que el sistema comience a trabajar. Una vez pulsado, tanto el botón como el cuadro de texto con la dirección y el selector del navegador se desactivarán.
- Un selector con la etiqueta --Selecciona el navegador a emular--: permite seleccionar al usuario qué navegador desea emular con HTMLUnit en la descarga de la página web que se va a analizar. Ofrece cuatro opciones: por defecto (es el que mejores resultados ofrece y por tanto se indica como recomendado), Chrome, Firefox e Internet Explorer. Hasta que no se seleccione uno de estos cuatro y la opción por defecto deja de aparecer, no será posible pulsar el botón aceptar.

- Un cuadro de texto con la etiqueta “Resultados”: en este cuadro se imprimirán los resultados del analizador y se dejará constancia por escrito de las dos probabilidades a partir de las cuales se toma la decisión. El cuadro, inicialmente en gris, se tornará verde si la decisión final es favorable (el código JavaScript no está ofuscado) y rojo, si por el contrario, es desfavorable (el sitio puede estar comprometido ya que el código JavaScript está ofuscado). Una vez finalizada la tarea para la que ha sido programado el sistema, el usuario tendrá capacidad para modificar, copiar o manipular de cualquier otra forma el texto que se muestra en los resultados.
- Una imagen: se puede apreciar a la derecha la imagen de una mano con un pulgar en horizontal. Como pasaba en las antiguas batallas entre gladiadores, esta mano deberá juzgar, tal y como lo hacía en su día el emperador romano, si el sitio web analizado puede seguir viviendo o debe morir, metafóricamente hablando. Básicamente, el pulgar pasará a apuntar hacia arriba si la decisión final del sistema es favorable, o hacia abajo si es desfavorable.
- Para concluir (finalizar el proceso), lo único que el usuario puede hacer es cerrar la ventana.

3.3. PRUEBAS

En esta sección se pretende poner a prueba el sistema desarrollado. Se harán diversas probaturas con webs utilizadas en el entrenamiento y otras que no hayan sido incluidas en el mismo, emulando diferentes navegadores. Las conclusiones que se saquen a raíz de esta fase de pruebas estarán recogidas en el apartado dedicado para ello, por lo que a continuación solo se encontrarán las URLs de los sitios analizados, el navegador emulado, una captura de la interfaz con los resultados que ofrezca el sistema y las observaciones oportunas cuando sean necesarias.

Prueba nº1

- URL: https://www.google.es/
- Navegador: Por defecto

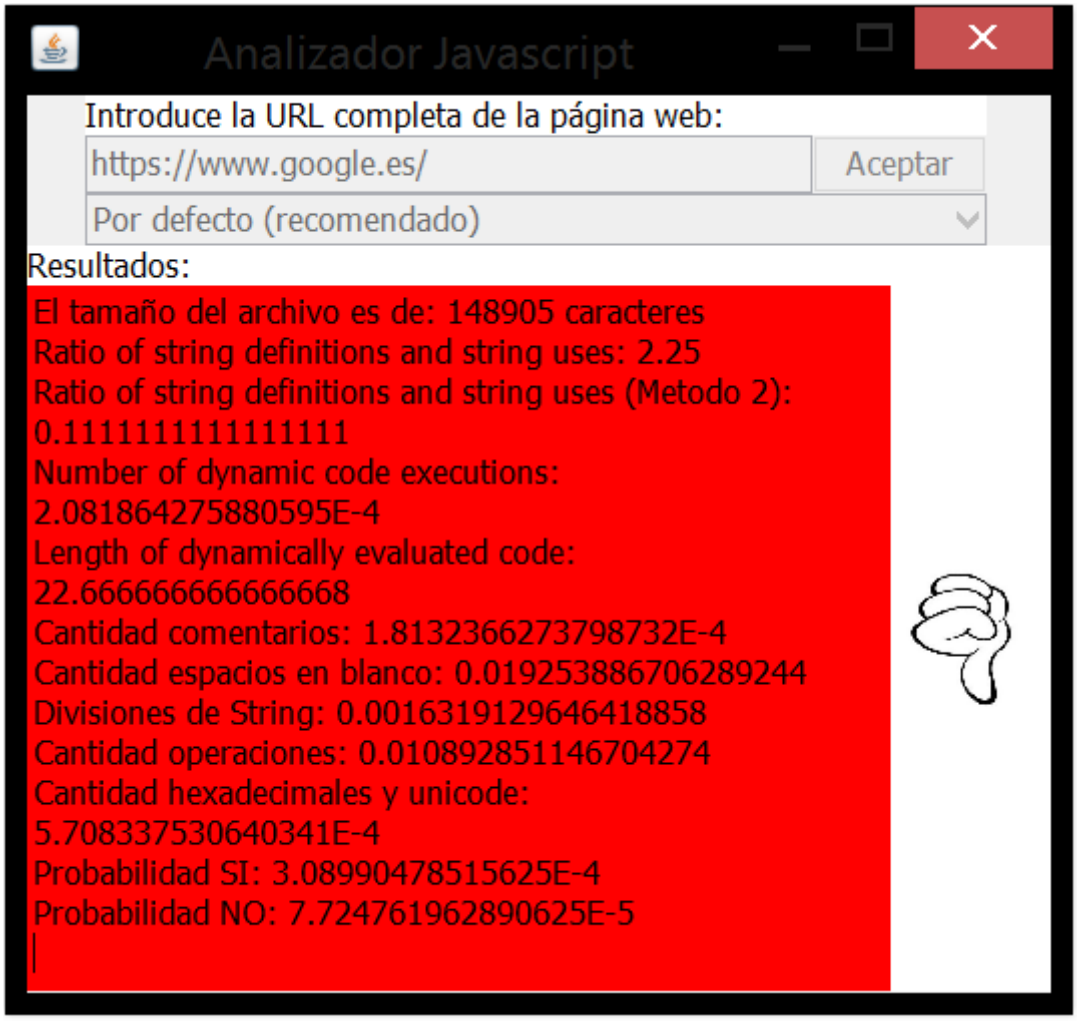

Figura 36: Resultados prueba nº1

Observaciones:

- No es posible descargar esta página desde un navegador que no sea Chrome o, como se aprecia en el ejemplo, el que utiliza HTMLUnit por defecto. Es algo que ocurre de forma bastante habitual y uno de los aspectos que se discutirán en las conclusiones. Notar que antes de dar por sentado que un enlace concreto no funciona, es conveniente probar con los cuatro navegadores disponibles para emular.
- El sistema determina que el buscador de Google en español puede

estar siendo comprometido por un atacante si atendemos a la ofuscación de código JavaScript. Aunque puede sorprender este resultado, es justamente lo que se esperaba, ya que como se ha explicado en varias ocasiones a lo largo del informe, la ofuscación es una técnica que habitualmente se aplica en páginas webs legítimas. Compañías de gran calado como Google, que manejan una cantidad de datos sensibles enorme, ofuscan el código masivamente.

Prueba nº2

- URL: https://es.wikipedia.org/
- Navegador: Chrome

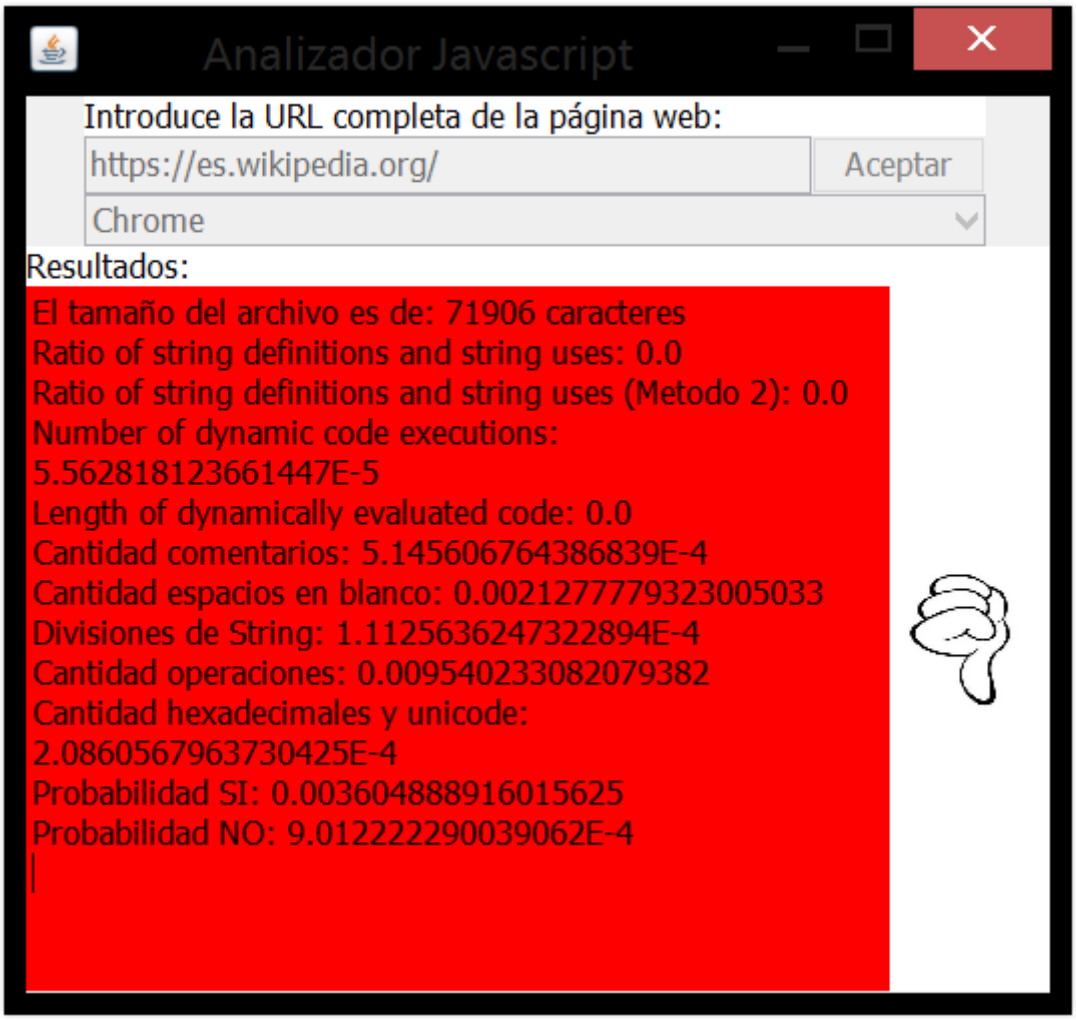

Figura 37: Resultados prueba nº2

Observaciones:

- Como ocurría con la prueba nº1, estos resultados no pueden ser obtenidos con cualquier navegador.
- Wikipedia también cuenta, según nuestro sistema, con código JavaScript ofuscado. Nuevamente y como ocurría con Google, concuerda el resultado esperado.

Prueba nº3

- URL: http://www.uv.es/jac/guia/jscript/reloj.htm
- Navegador: Firefox

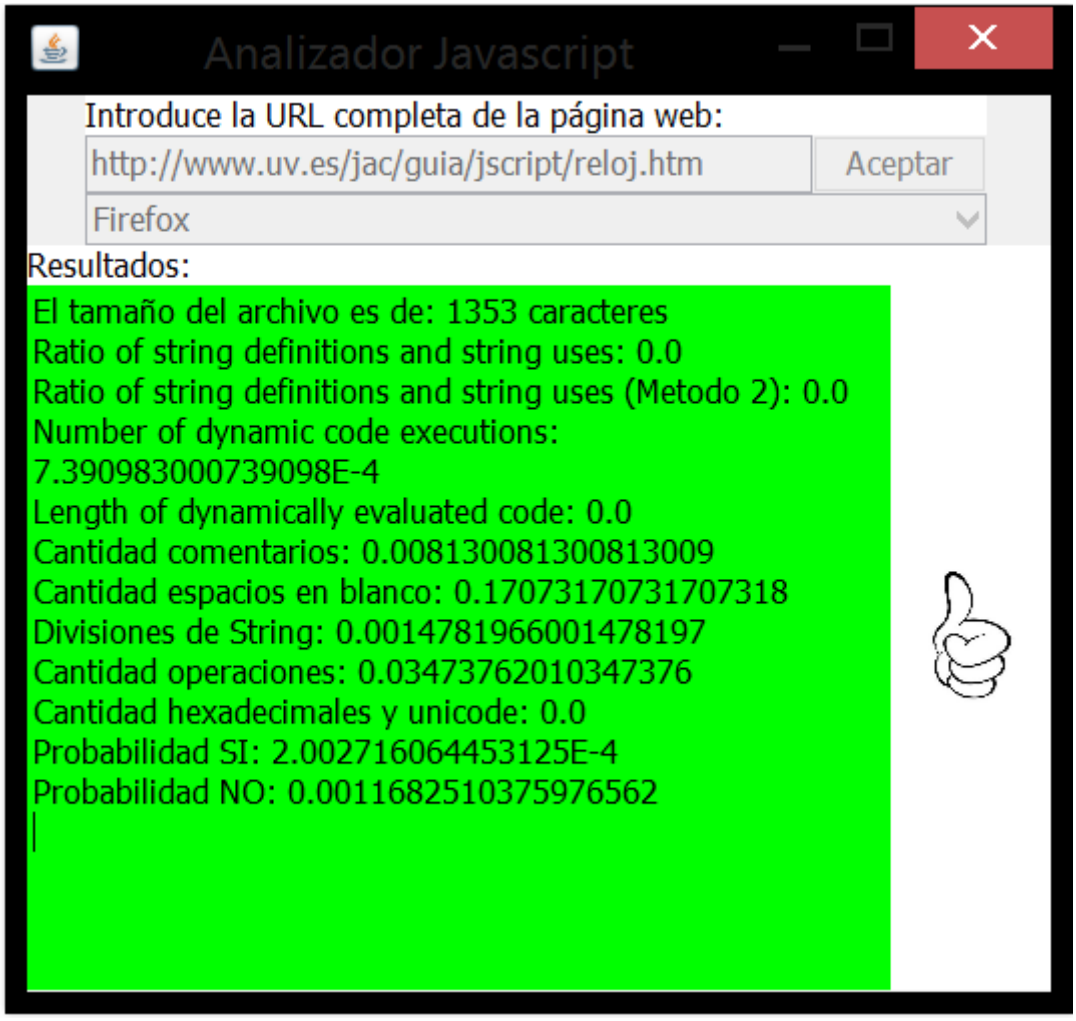

Figura 38: Resultados prueba nº3

Observaciones:

- Por motivos que se desconocen, estas URLs, correspondientes con

algunas de las muestras utilizadas en el entrenamiento, empezaron a dar error y el sistema solo era capaz de analizarlas si introducíamos la dirección precedida de *http://* en lugar de *https://* por lo que es conveniente apuntar este como uno de los posibles errores.

- Como era de esperar, el sistema determina que esta web no está en riesgo de ser atacada atendiendo a la ofuscación del código JavaScript. Se puede comprobar fácilmente que estos códigos, efectivamente, no están ofuscados, ya que son meros ejemplos.

Prueba nº4

- URL: http://www.uv.es/jac/guia/jscript/calendario.html
- Navegador: Internet Explorer

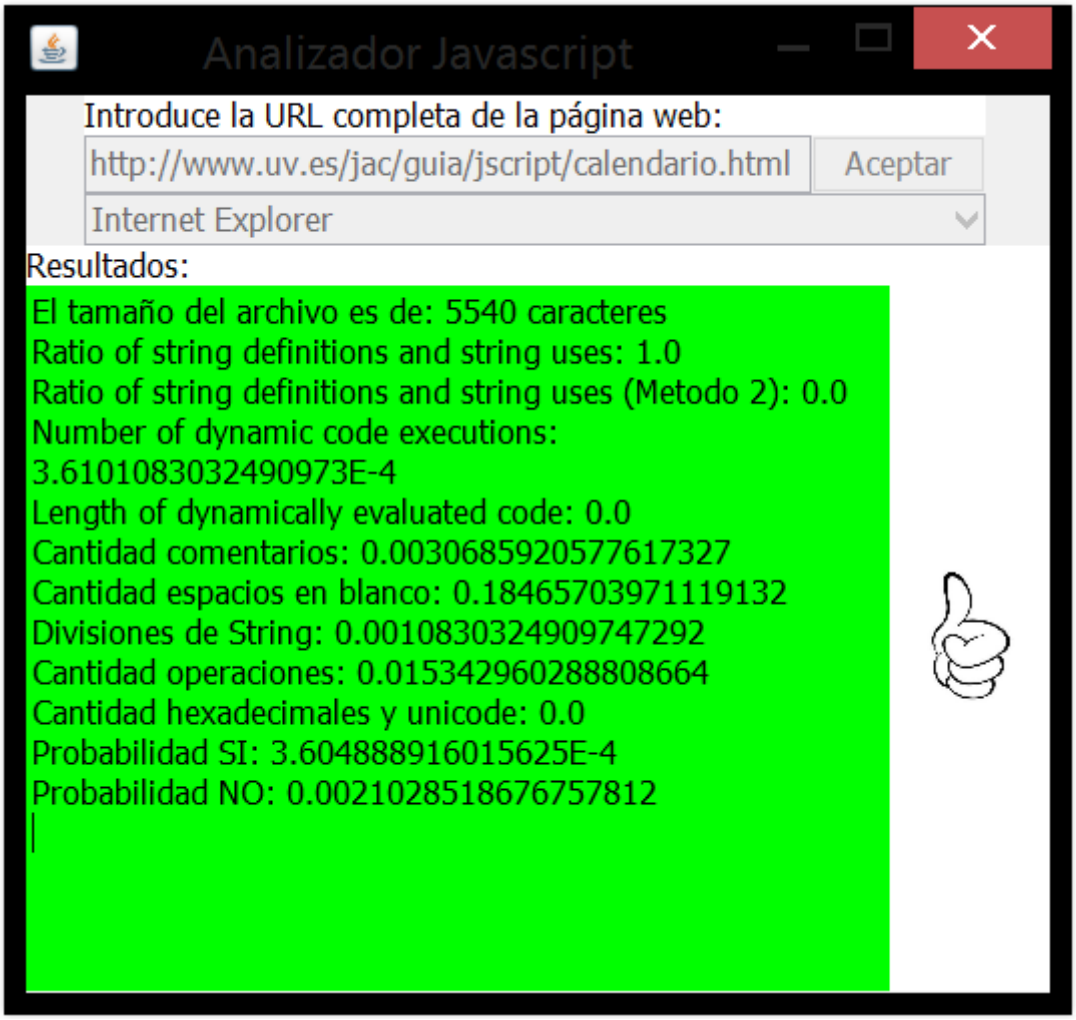

Figura 39: Resultados prueba nº4

Observaciones:

- Un nuevo ejemplo de esta web, mismo resultado: el código JavaScript no está ofuscado.
- Notar que estas páginas pueden ser analizadas emulando cualquiera de los navegadores disponibles, lo que se puede interpretar como que Firefox e Internet Explorer resultan más problemáticos a medida que aumenta la complejidad de la página a analizar.

Prueba nº5

- URL: https://www.facebook.com/
- Navegador: Por defecto

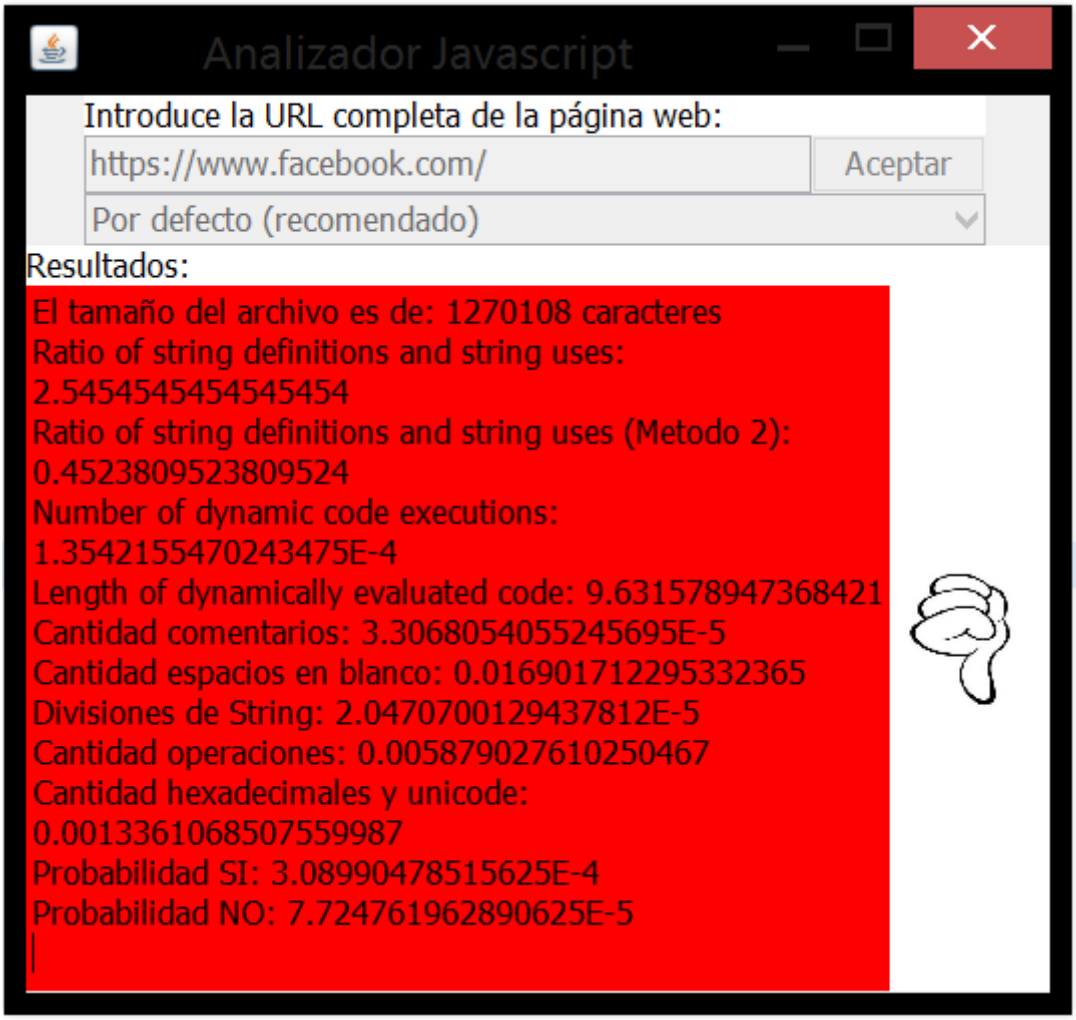

Figura 40: Resultados prueba nº5

Observaciones:

- Facebook, otra web de una gran compañía, mismo resultado y mismo problema de compatibilidad con algunos navegadores.

Prueba nº6

- URL: http://www.anaitgames.com/
- Navegador: Chrome

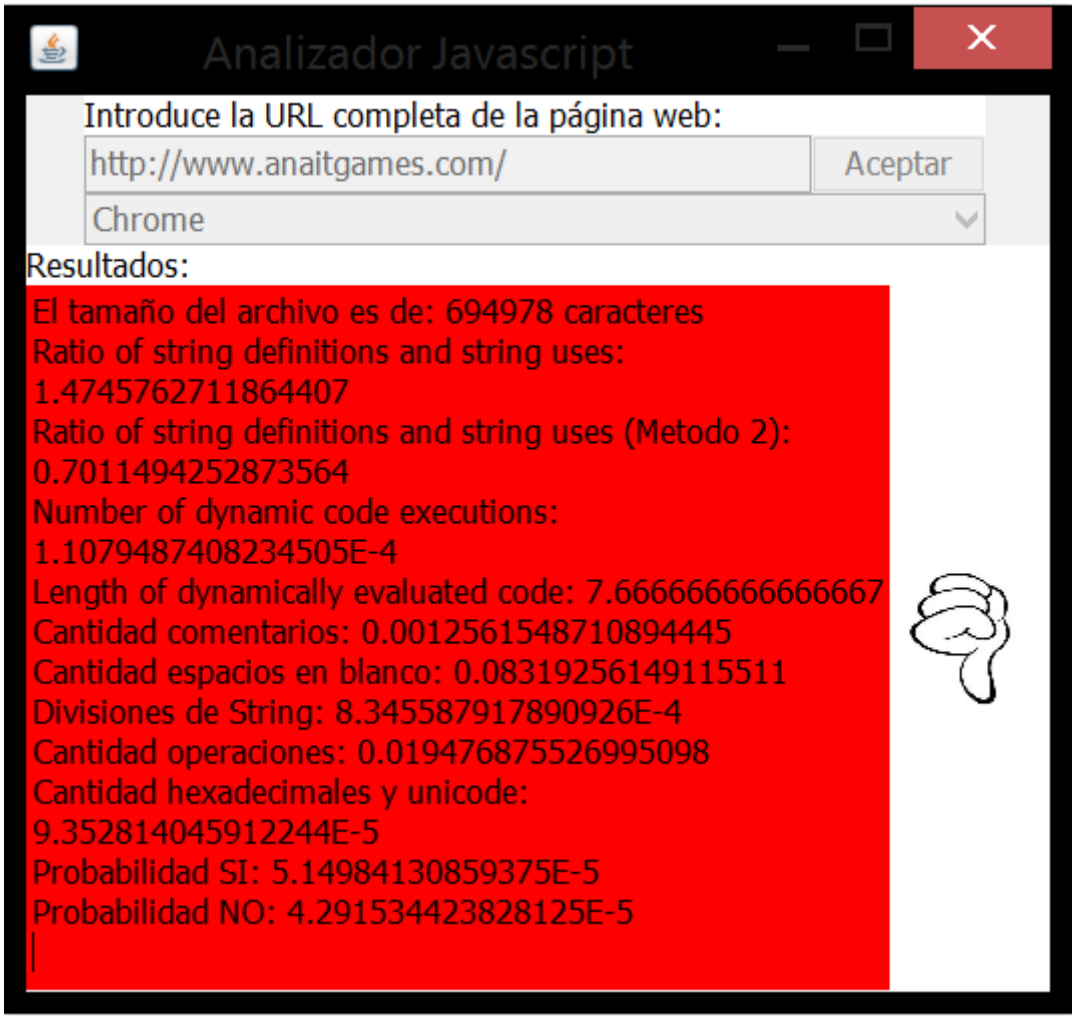

Figura 41: Resultados prueba nº6

Observaciones:

- AnaitGames es una página web dedicada al mundo de los videojuegos. Aunque no es propiedad de una gran compañía y su impacto no es comparable por ejemplo al de Facebook, sí que parte de la información que utiliza, incluidos usuarios y contraseñas de las personas registradas, es sensible y por tanto se han aplicado, según

nuestro sistema, algunas técnicas de ofuscación sobre el código JavaScript.

Prueba nº7

- URL: https://swad.ugr.es/
- Navegador: Firefox

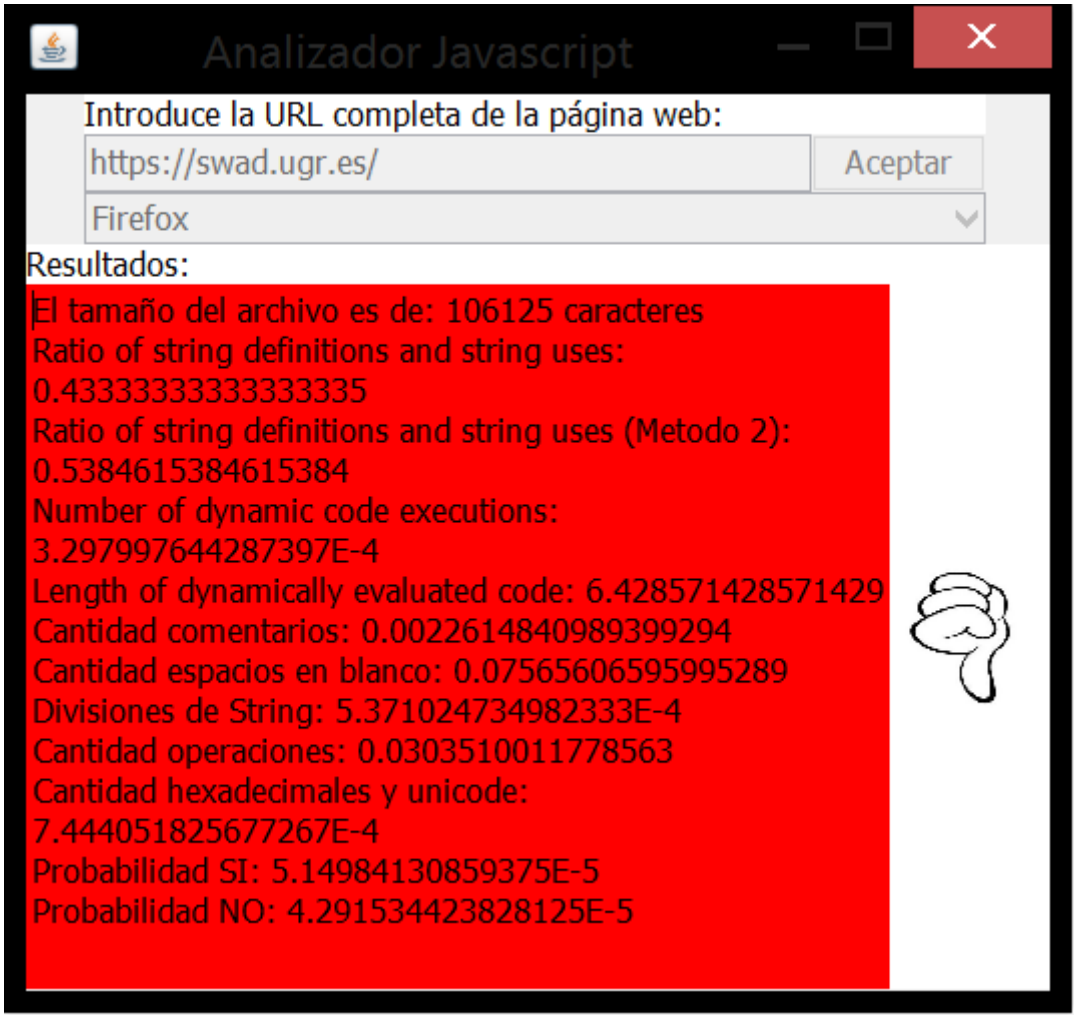

Figura 42: Resultados prueba nº7

Observaciones:

- Poco que comentar en este caso salvo que el resultado vuelve a ser desfavorable (lo esperado) y que Firefox sí que funciona.

Prueba nº8

- URL: http://htmlunit.sourceforge.net/
- Navegador: Internet Explorer

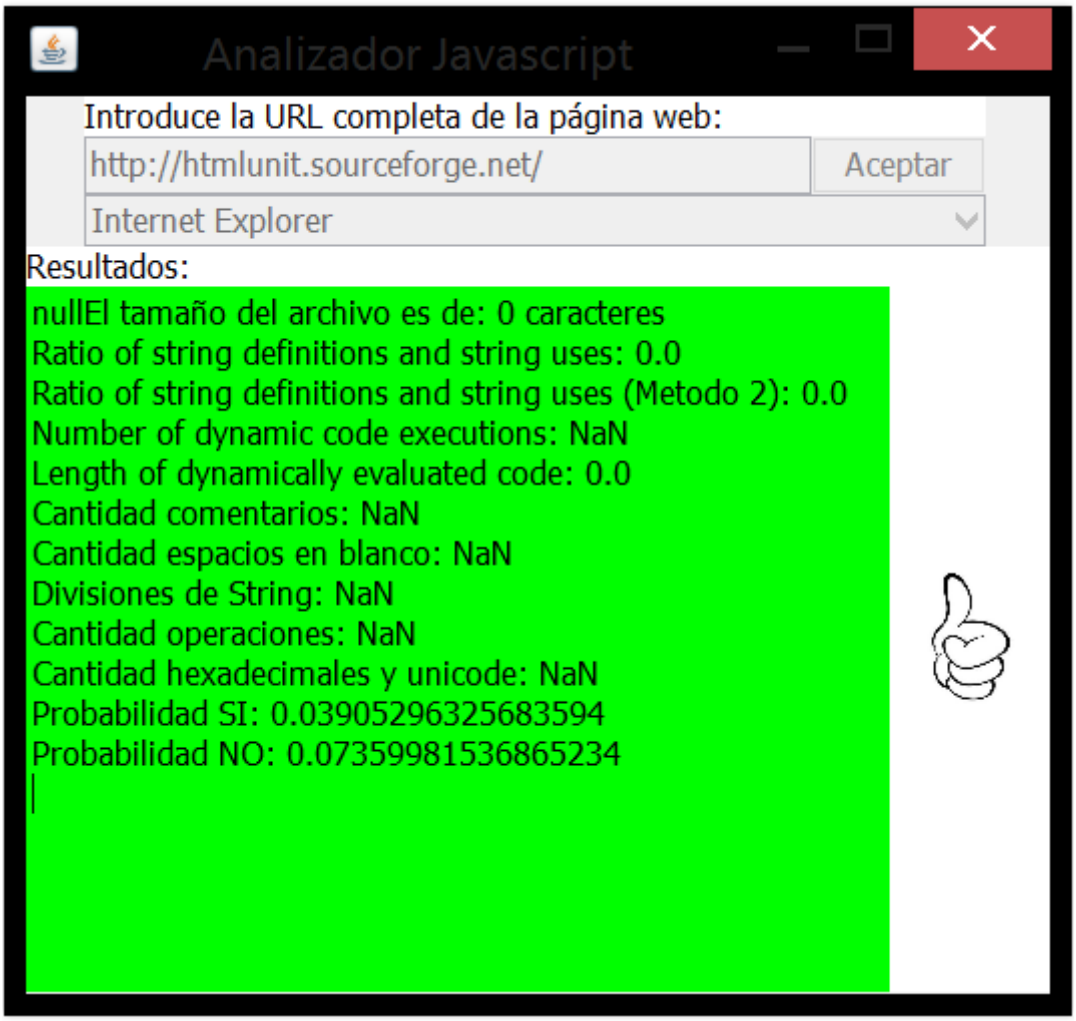

Figura 43: Resultados prueba nº8

Observaciones:

- La web de inicio de HTMLUnit es posiblemente uno de los ejemplos más interesantes. Se puede apreciar como los resultados son 0 o nulos para la mayoría de características ya que este sitio no ejecuta código JavaScript.
- Lo realmente importante es que nuestro sistema, dado un caso tan extremo como que directamente no existe código JavaScript para analizar, resuelve de forma acertada y determina que el sitio no está comprometido.

4. CONCLUSIONES Y TRABAJO FUTURO

Este último capítulo estará dedicado a exponer las principales conclusiones extraídas del proceso de elaboración del presente proyecto. Se desarrollará una valoración personal y se detallarán aspectos como el nivel de satisfacción con los resultados logrados, los principales obstáculos que se han encontrado y finalmente se indicarán posibles mejoras para el sistema implementado de cara a que un trabajo posterior pueda profundizar aún más en la idea de detectar ataques Drive-by-Download centrándose en la ofuscación de código JavaScript.

4.1. VALORACIÓN PERSONAL

Desde la perspectiva individual del autor, Lucas Cruz, este proyecto ha supuesto una grata experiencia, gracias en gran medida al apoyo y la disposición del tutor, Gabriel Maciá, a la par que un gran reto, no solo en al ámbito académico sino también en lo personal, especialmente si se tienen en cuenta los condicionantes externos que han afectado al avance del trabajo.

Por todo esto, se puede decir que resulta altamente gratificante haber logrado, finalmente con éxito, la finalización del proyecto. Nunca antes me había enfrentado a un proyecto de una magnitud similar, con multitud de conceptos nuevos que aprender, la utilización de herramientas que nunca antes habían pasado por mis manos ni de las de prácticamente ningún compañero del grado, y una duración prolongada expuesta a posibles inconvenientes que se han de saber solucionar para alcanzar la meta.

Mentiría si dijera que ha sido un camino de rosas, pero de todo se aprende y sin duda, este proyecto me ha enseñado algunas de las lecciones más valiosas de entre todas las recibidas en los últimos años como estudiante. Conocimiento tanto del mundo al que nos enfrentaremos en breve, como de uno mismo, que seguro será de gran utilidad para mi desarrollo como profesional en los años venideros.

4.2. CONCLUSIONES

Desde el punto de vista de las habilidades desarrolladas:

- Se ha asimilado el funcionamiento de los ataques Drive-by-Download lo que ha servido para entender mejor cómo funciona la mente de los atacantes, algo que se puede extrapolar a cualquier amenaza a la que nos enfrentemos en un futuro.
- Se han estudiado y entendido cómo funcionan las principales técnicas de ofuscación, cuáles son las más utilizadas, por qué es algo de especial relevancia actualmente si se habla en términos de seguridad en Internet, sus posibles usos tanto por web master legítimos como por posibles atacantes, etcétera.
- Se ha logrado dominar una herramienta compleja como HTMLUnit. Si bien sus posibilidades son mucho más amplias de las aquí utilizadas, la experiencia adquirida debería ser más que suficiente para su aplicación en futuros proyectos.
- Se han mejorado las habilidades de programación en el lenguaje Java. Aunque durante el grado se han cursado varias asignaturas que han abordado la enseñanza de 'la programación', alguna incluso con Java como lenguaje central, durante los meses que han transcurrido desde que se comenzara a trabajar se han aprendido algunos nuevos conceptos. De especial interés el cariz 'más profesional' ya que el sistema desarrollado no se limita a la obtención de una nota para un ejercicio, sino que debe tener una lógica estructural, ser entendible por otras personas, dejar abierta la puerta a posibles mejoras… aspectos fundamentales cuando se trabaja en una empresa/organización.
- Se ha profundizado en ideas como la planificación, el establecimiento de objetivos, la organización de los agentes implicados, la financiación, la gestión de recursos o la documentación de un proyecto.

Desde el punto de vista de los resultados obtenidos:

- El sistema es capaz, en una amplia mayoría de los casos, de extraer el código JavaScript que ejecuta la página que se desea analizar. Cuando no ocurre es debido a que HTMLUnit entra en conflicto con algunos elementos dispuestos para proteger la integridad de los sitios web. La solución de este problema se plantea como uno de los aspectos a mejorar en el futuro.
- El sistema analiza hasta ocho características distintivas de los códigos ofuscados. Son más de las que se habían planteado en un principio, y menos de las que se podrían implementar. Si bien, teniendo en cuenta los objetivos del proyecto, el tiempo disponible y los recursos al alcance, son más que suficientes para ofrecer un resultado satisfactorio a todos los niveles.
- El sistema ofrece los resultados esperados prácticamente en el 100% de las pruebas realizadas, y así se refleja en la sección dedicada. Desgraciadamente no podemos comprobar en todos los casos si nuestro sistema acierta realmente o no, pero cuando es posible, la fiabilidad es total, incluso en casos extremos.
- Siguiendo con el punto anterior, tanto el entrenamiento como el clasificador diseñado cumplen a la perfección con su cometido. A pesar que por las limitaciones existentes han impedido tanto la realización de un entrenamiento mucho más amplio como un estudio en mayor profundidad de los clasificadores probabilísticos que se podían utilizar, el resultado final es prácticamente inmejorable.
- Se ha ejecutado con éxito la implementación de un sistema modular donde cada módulo cumpla con su función de forma individual. Este es un requisito indispensable si se quiere, como se había planteado en primera instancia, dejar abierta la posibilidad para que este sistema sea mejorado en cualquiera de sus fases y cómo no, pueda ser integrado como parte de un sistema mayor que se enfrente de manera íntegra a un ataque Drive-by-Download.
- La interfaz también cumple con el objetivo asignado, acercando el

sistema a cualquier usuario que desee comprobar si una web puede estar comprometida atendiendo la ofuscación del código JavaScript. De la misma forma, se han incorporado elementos visuales que facilitan la lectura de los resultados.

4.3. LÍNEAS DE DESARROLLO FUTURO

A pesar del alto grado de satisfacción que se ha alcanzado con el desarrollo de este proyecto, existen algunos aspectos que se podrían mejorar en el futuro.

- Solucionar los problemas con HTMLUnit: hay páginas webs que entran en conflicto con HTMLUnit cuando tratamos de extraer el código JavaScript. La mayoría de veces se soluciona cambiando el navegador emulado o quitando/añadiendo la 's' de *https,* pero en algunos casos no termina por funcionar. Para solucionar este problema habría que examinar en profundidad todos los métodos de la librería HTMLUnit utilizados y modificar las líneas necesarias en caso de detectar dónde se produce el conflicto.
- Incorporar nuevas características: aunque ocho son suficientes para lograr el objetivo marcado en este proyecto, podrían añadirse nuevas características para analizar en el segundo módulo, lo que podría dar pie presuntamente a una mejora de los resultados.
- Selección de muestras más apropiada: la fase de enteramiento ha sido uno de los puntos donde más problemas se han encontrado. Las muestras utilizadas se han obtenido con el primer módulo del sistema cuando lo ideal sería que se dispusiera de antemano de un banco de pruebas integrado por número amplio de códigos JavaScript con características conocidas, entre los que se encontraran códigos no ofuscados de tamaño considerable, códigos ofuscados sin intención de atacar y códigos ofuscados con intención de ocultar las instrucciones de un ataque Drive-by-Download. De

ser así se podrían explorar con mayor conocimiento de causa las diferencias entre código ofuscado y no ofuscado, incluso cabría la posibilidad de determinar si la ofuscación ha sido realizada por un delincuente o no. Obtener este muestrario ha resultado imposible con los medios disponibles.

- Utilización de herramientas de ofuscación complejas: se han utilizado cuatro de las muchas herramientas de ofuscación disponibles en Internet, y es que ciertamente, los resultados del entrenamiento no variaban con la incorporación de más herramientas gratuitas. En un futuro y con un presupuesto mayor, sería interesantes ampliar el número de herramientas utilizadas con algunas de pago, incluso con alguna que se pudiera obtener gracias a alguien que tenga contactos con alguno de estos atacantes o haya sido descubierta gracias a la acción policial.
- Estudio de los clasificadores probabilísticos: la selección del clasificador Naive Bayes se debe a que cumple con la mayoría de requisitos impuestos para sistema desarrollado, lo que no quita que pueda haber otro que todavía se ajuste mejor a las necesidades del mismo. Un experto en la materia podría realizar un estudio y dictaminar si existe uno mejor así como realizar su correspondiente diseño.
- Otros añadidos: se podrían pulir aspectos como la interfaz o tener en consideración todos los posibles errores de tal manera que el sistema no se detenga y siga funcionando mostrando el correspondiente mensaje en caso de que ocurra un error.

5. APÉNDICE: REPOSITORIO DE CÓDIGO

En el apartado 4.3. se han definido cuales deberán ser las líneas de trabajo futuras para cualquiera que esté interesado en profundizar en la idea que se propone en este proyecto y/o mejorar el sistema que se ha implementado durante el tiempo que ha durado su desarrollo.

A fin de facilitar esta tarea se propone la creación de un repositorio en GitHub que permita el acceso al material que ha resultado de dicho desarrollo a todo lector de este informe que desee continuar con el propósito de crear un sistema capaz de detectar ataques Drive-by-Download, en especial para aquel que se sienta atraído por las técnicas de ofuscación involucradas, su análisis, caracterización y distinción del código ofuscado del código sin ofuscar. El código se puede encontrar en el siguiente enlace:

https://github.com/LucasCruzC/TFG-Lucas-Cruz

Se pueden descargar cada una de las piezas de código por separado o si se prefiere, el conjunto de todas ellas en formato ZIP. Se ha intentado comentar el código en la medida de lo posible con el objetivo de que junto al estudio de este informe se pueda conocer para qué sirve cada una de las instrucciones presentes. Están incluidas no solo las piezas de código que forman el sistema final que ha resultado de este proyecto sino también aquellas que se han utilizado en alguna de las fases previas.

5.1. GITHUB

Figura 44: Logo GitHub

GitHub es una plataforma de desarrollo colaborativo que permite alojar proyectos utilizando el sistema de control de versiones Git. Este sistema de control de versiones facilita la gestión de forma eficiente de cada cambio que se efectúa sobre los diferentes ficheros que conforman el proyecto, en este caso, los diferentes fragmentos de código.

La plataforma actúa de sustento para proyectos de todo tipo, desde Wikis (sitios web cuyas páginas pueden ser editadas desde el navegador, donde los usuarios crean, modifican o eliminan contenidos), juegos web, contenidos literarios (los autores pueden compartir sus obras) o realidad virtual, una de las tecnologías más prometedoras a día de hoy a unos años vista. Si bien, es cierto que entre todos estos, GitHub dispone de una particular popularidad entre los desarrolladores de software libre, siendo partícipe de proyectos de gran envergadura como JQuery o reddit, entre muchos otros.

Aunque GitHub cuenta con una versión de pago para particulares y empresas, que permite en líneas generales la creación de repositorios privados y la interacción entre usuarios en diferentes proyectos, la versión gratuita ofrece una retahíla de características más que suficiente para considerar esta como la plataforma idónea a la hora de compartir el código implementado como parte del desarrollo del trabajo propuesto. Entre otras, dispone de:

- Un visor de código que permite a través del navegador consultar rápidamente el contenido de un determinado fichero con la sintaxis correspondiente para el lenguaje en el que esté escrito.
- Un sistema de seguimiento de problemas mediante tickets.
- Una herramienta de revisión de código que facilita la adición de anotaciones en cualquier punto de un fichero.
- Un visor de ramas (branches) donde se pueden comprobar los cambios realizados y la evolución seguida por las distintas ramas del repositorio.
- Trabajo colaborativo que permite clonar repositorios ajenos (fork), la modificación de los ficheros y posterior solicitud (pull) al dueño original que puede aceptar o rechazar los cambios propuestos.

6. REFERENCIAS

[1] Marco Cova, Christopher Kruegel, and Giovanni Vigna. Detection and Analysis of Drive-by-Download Attacks and Malicious JavaScript Code.

[2] Aikaterinaki Niki (2009). Drive-by-Download attacks: effects and detection methods.

[3] Wei Xu, Fangfang Zhang and Sencun Zhu. The Power of Obfuscation Techniques in Malicious JavaScript Code: A Measurement Study.

[4] Zorabedian, J. (2014). How malware works: Anatomy of a drive-by download web attack (Infographic) | Sophos Blog. [online] Blogs.sophos.com. Disponible en: https://blogs.sophos.com/2014/03/26/how-malware-works-anatomy-of-a-drive-by-download-web-attack-infographic/ [Accesible 18-Febrero-2016].

[5] Bowler, M., Guillemot, M. and Ashour, A. (2016). HtmlUnit – Welcome to HtmlUnit. [online] Htmlunit.sourceforge.net. Disponible en: http://htmlunit.sourceforge.net/ [Accesible 29-Febrero-2016].

[6] Eclipse.org. (2016). [online] Disponible en: https://eclipse.org/ [Accesible 21-Marzo-2016].

[7] Junit.org. (2016). JUnit. [online] Disponible en: http://junit.org/ [Accesible 02-Marzo-2016].

[8] Oracle.com. (2016). Software Java | Oracle España. [online] Disponible en: https://www.oracle.com/es/java/index.html [Accesible 02-Junio-2016].

[9] Scribd. (2016). Características del lenguaje Java VENTAJAS Y DESVENTAJAS. [online] Disponible en: https://es.scribd.com/doc/165321281/Caracteristicas-del-lenguaje-Java-VENTAJAS-Y-DESVENTAJAS [Accesible 02-Junio-2016].

[10] Text Analysis blog | Aylien. (2015). Naive Bayes for Dummies; A Simple Explanation. [online] Disponible en: http://blog.aylien.com/post/120703930533/naive-bayes-for-dummies-a-simple-explanation [Accesible 25-Abril-2016].

[11] Inf.u. (2016). [online] Disponible en: http://www.inf.u-szeged.hu/~ormandi/ai2/06-naiveBayes-example.pdf [Accesible 25-Abril-2016].

[12] Es.wikipedia.org. (2016). JavaScript. [online] Disponible en: https://es.wikipedia.org/wiki/JavaScript [Accesible 5-Junio-2016].

[13] Es.wikipedia.org. (2016). Java (lenguaje de programación). [online] Disponible en: https://es.wikipedia.org/wiki/Java_(lenguaje_de_programaci%C3%B3n) [Accesible 5-Junio-2016].

[14] Es.wikipedia.org. (2016). MATLAB. [online] Disponible en: https://es.wikipedia.org/wiki/MATLAB [Accesible 5-Junio-2016].

[15] Es.wikipedia.org. (2016). HTML. [online] Disponible en: https://es.wikipedia.org/wiki/HTML [Accesible 5-Junio-2016].

[16] Es.wikipedia.org. (2016). Document Object Model. [online] Disponible en: https://es.wikipedia.org/wiki/Document_Object_Model [Accesible 5-Junio-2016].

[17] Htmlunit.sourceforge.net. (2016). HtmlUnit 2.22 API. [online] Disponible en: http://htmlunit.sourceforge.net/apidocs/index.html?com/gargoylesoftware/htmlunit/ [Accesible 29-Febrero-2016].

[18] GitHub. (2016). Build software better, together. [online] Disponible en: https://github.com/ [Accesible 20-Junio-2016].

[19] Paramio, C. (2011). Conociendo GitHub, el servicio donde alojar tus repositorios Git (como el nuestro). [online] Genbetadev.com. Disponible en: http://www.genbetadev.com/sistemas-de-control-de-versiones/conociendo-github-el-servicio-donde-alojar-tus-repositorios-git-como-el-nuestro [Accesible 20-Junio-2016].