HTB - Challenge: Hypercraft	
Dificultad:	**Medium**
Release:	**06/07/2023**

Skills Required

- Basic deobfuscation techniques
- Basic JavaScript

Skills Learned

- JavaScript deobfuscation
- Powershell deobfuscation
- Windows task scheduler persistence techniques

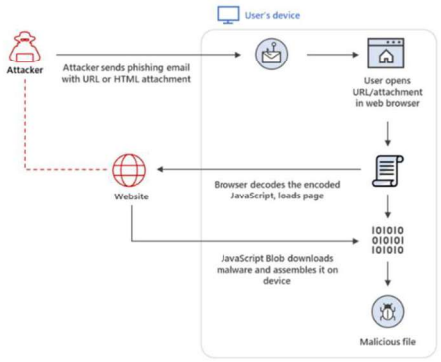
En el contexto de mis actividades de capacitación avanzada en ciberseguridad ofensiva y análisis de artefactos maliciosos, he llevado a cabo una evaluación técnica del laboratorio **Hypercraft** de Hack The Box. El ejercicio reproduce un escenario de amenaza verosímil, articulado en torno a una cadena de infección basada en *HTML smuggling*, reconstrucción progresiva de payloads en cliente y ejecución secuencial de componentes ofuscados mediante JavaScript y PowerShell. Su estructura lo convierte en un entorno idóneo para examinar técnicas contemporáneas de evasión, mecanismos anti-análisis y patrones de distribución empleados en campañas reales de malware.

El presente informe documenta de forma rigurosa el proceso de análisis, desofuscación y reconstrucción de cada uno de los estadios del payload, desde la inspección inicial del correo electrónico y su adjunto hasta la obtención del artefacto final. Se detallan las metodologías aplicadas, los indicadores observados y las decisiones analíticas adoptadas en cada fase, con el objetivo de ofrecer una visión clara, sistemática y alineada con las prácticas profesionales de *threat analysis*, *malware triage* y *incident response*. El propósito de este documento es evidenciar un enfoque metódico, técnicamente solvente y plenamente extrapolable a entornos corporativos donde la detección temprana y la comprensión profunda de vectores de ataque resultan críticas para la mitigación del riesgo.

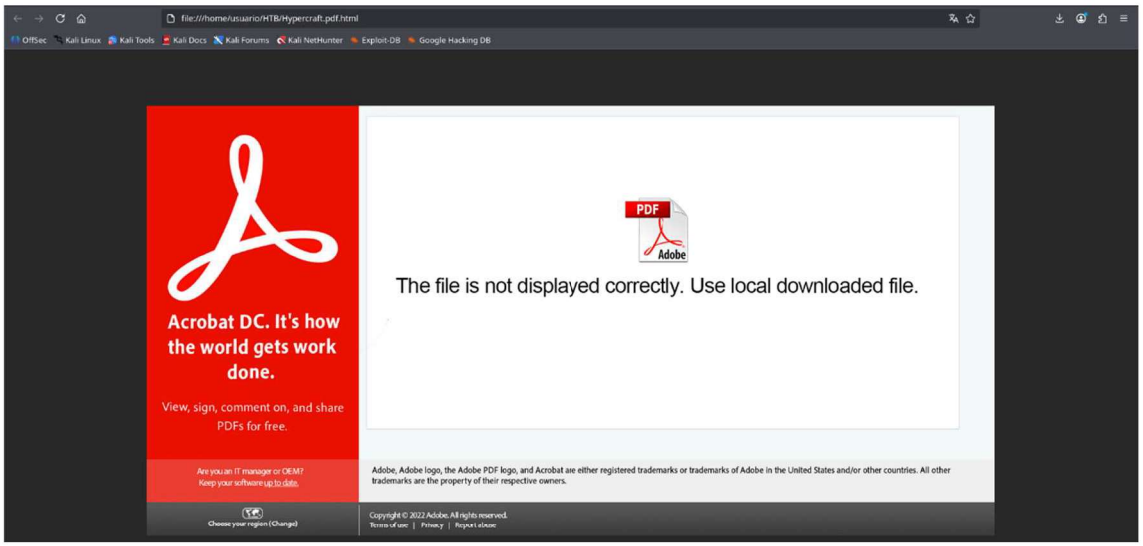
Enumeración

El análisis inicial parte del fichero **.cml**, cuyo contenido revela un mensaje redactado con un marcado tono de urgencia acompañado de un archivo adjunto. Este archivo, identificado como **[TOP SECRET] Arodorian Hypercraft.pdf.html**, constituye el primer vector de interés. Tras extraerlo y examinarlo, se comprueba que se trata de una página HTML camuflada bajo una extensión doble, una técnica habitual en campañas de ingeniería social orientadas a la ejecución inadvertida de contenido activo.

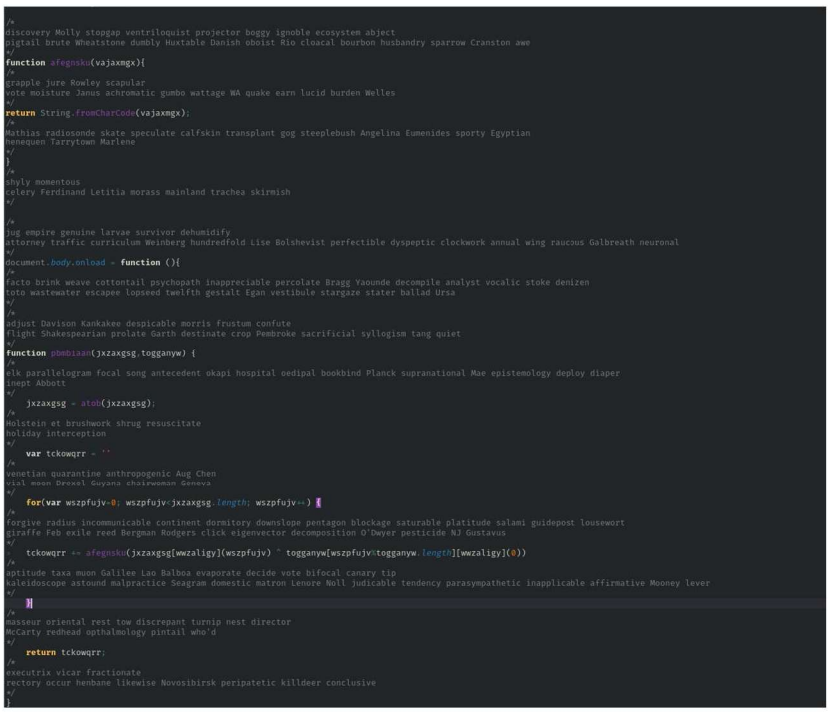
Al abrir el archivo en un navegador aislado —en este caso, Firefox— la página simula un error de Adobe Reader y, de manera automática, desencadena la descarga de un archivo comprimido en formato ZIP. Este comportamiento confirma la presencia de una cadena de infección basada en **HTML smuggling**, un método de distribución de payloads que aprovecha la capacidad del navegador para reconstruir artefactos binarios a partir de datos embebidos en el propio documento HTML.



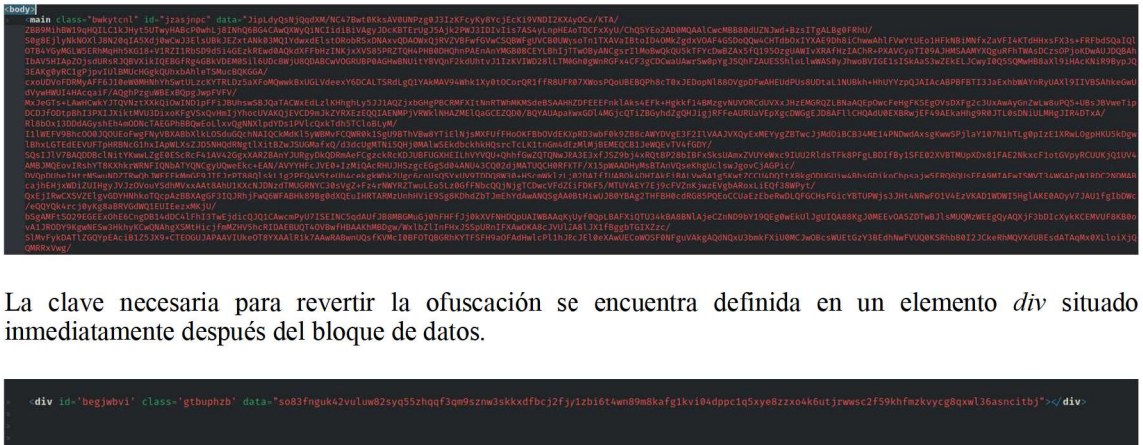
Esta técnica permite eludir mecanismos perimetrales de inspección, ya que el fichero malicioso no se transfiere como un binario convencional, sino que se genera localmente en el cliente mediante JavaScript, codificación Base64 u otros mecanismos de ofuscación. Como consecuencia, los controles de seguridad basados en análisis de tráfico o filtrado de adjuntos quedan sustancialmente debilitados.



El código fuente del HTML contiene varios bloques extensos de datos codificados en **Base64**, intercalados con comentarios superfluos que actúan como ruido para dificultar el análisis estático. Entre las funciones presentes destaca **pbmbiaan**, cuyo propósito consiste en recibir dos cadenas, decodificar la primera desde Base64 y aplicar posteriormente una operación XOR con la segunda. Este patrón es característico de mecanismos rudimentarios de ofuscación empleados para evadir firmas y retrasar la inspección manual del contenido malicioso.



El contenido ofuscado presente en el documento HTML no solo reconstruye la interfaz fraudulenta —incluida la imagen simulada de Adobe Reader en segundo plano—, sino que también genera dinámicamente el archivo ZIP que se descarga de manera automática al abrir la página. El bloque de datos correspondiente al ZIP aparece en la parte superior del código fuente, encapsulado en una cadena codificada en Base64 que constituye el núcleo del payload.

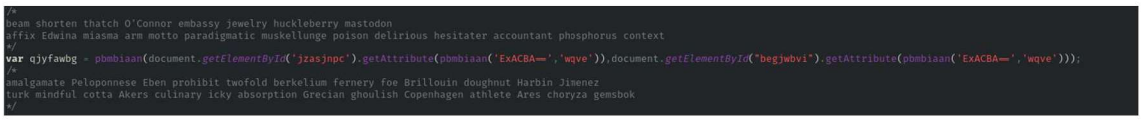


La clave necesaria para revertir la ofuscación se encuentra definida en un elemento *div* situado inmediatamente después del bloque de datos.

En la línea 132 del documento, ambas piezas —la cadena codificada y la clave— se combinan mediante la función previamente identificada, que decodifica el contenido Base64 y aplica una operación XOR para obtener el binario final. Este mecanismo ejemplifica una estrategia de **client-side payload reconstruction**, un enfoque en el que el navegador del usuario actúa como entorno de ensamblado del artefacto malicioso.

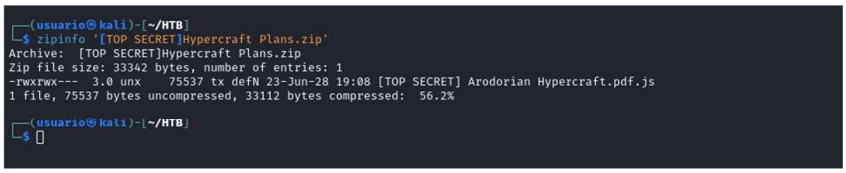
En lugar de transferir un binario completo a través de la red —lo que facilitaría su detección por soluciones de inspección perimetral— el atacante fragmenta y ofusca el payload dentro del propio HTML, delegando en el motor de JavaScript la tarea de recomponerlo localmente. Esta técnica erosiona la eficacia de los controles de seguridad basados en análisis de tráfico, ya que el fichero malicioso no existe como tal hasta que el navegador lo materializa en el sistema de la víctima.

A efectos prácticos, el análisis puede abordarse de dos maneras: replicando manualmente el proceso de decodificación a partir del código HTML o, de forma más directa, capturando el archivo ZIP que el navegador descarga automáticamente al renderizar la página. Ambas aproximaciones conducen al mismo artefacto, si bien la segunda permite agilizar la fase inicial del análisis sin comprometer la rigurosidad metodológica.



[TOP SECRET] Arodorian Hypercraft.pdf.js

El archivo ZIP obtenido en la fase anterior contiene un único fichero denominado, cuya extensión pretende reforzar la ilusión de legitimidad aludiendo a un supuesto documento PDF, aunque en realidad se trata de un script JavaScript ofuscado.

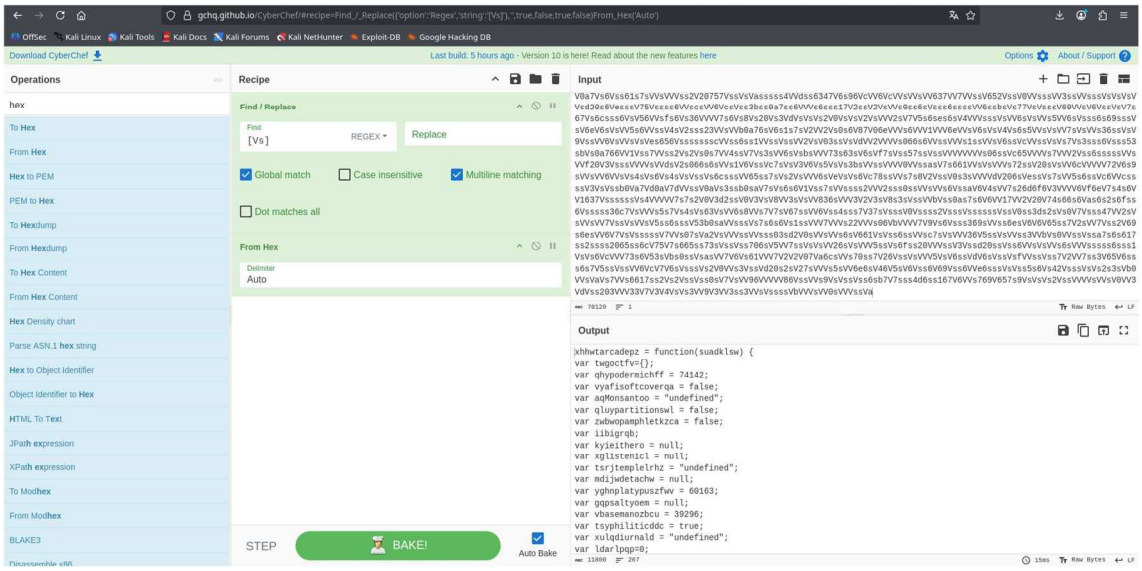


El análisis preliminar revela cuatro bloques textuales de gran tamaño acompañados de un conjunto reducido de funciones auxiliares. Estos bloques son concatenados para generar un flujo unificado de caracteres que constituye la base del siguiente estadio de ejecución.

El script incorpora un bucle infinito que selecciona de manera reiterada un número aleatorio entre 1 y 1000, avanzando únicamente cuando el valor coincide con **532**. Este artificio no aporta ninguna funcionalidad real, pero actúa como mecanismo de *anti-analysis*, introduciendo retardos artificiales destinados a entorpecer la ejecución en entornos instrumentados o a frustrar la depuración manual.

```
/*
rebellion fibration graham longitudinal grail exhume runty percent incurred
workshop pecan establish Shattuck trifle Katie
*/

while (hfhwsgmb > 0) {

/*
Laos humiliate Erwin opposition hygroscopic excelsior rain harmonic treble
grant quaint citadel Miltonic degrease phrase fusty alpaca bang
*/

hfhwsgmb = Math.floor(Math.random() * 10000)+1;

/*
Oscar Atkins Kresge Northrup furrier
turn autochthonous archaism tempo Valerie lens
*/

switch (hfhwsgmb)

/*
yesteryear cajole haggard motorcar badminton woodcarver entendre Cottrell apology
torpedo Starkey Teflon make colorimeter
*/

{

/*
Yuki saint inception skat intestate jetliner
raincoat adipic beckon Aristotelean Krakow aftermath saloon exploratory
*/

case 532:

/*
drake carcinoma ceil Lenore Solomon Lenten Parsi frankfurter
globular gripe tussle equilibrium crane Celtic moneywort cowhide
*/

hfhwsgmb = uwetjyhi.replace(/[sV]/g,'');
```

Una vez superado este bloqueo artificial, el código procede a depurar la cadena resultante eliminando los caracteres **s** y **V**, para posteriormente interpretar el flujo restante como una secuencia hexadecimal. Tras convertirla a binario, el script ejecuta dinámicamente el resultado mediante una invocación directa al motor de JavaScript. Este proceso puede reproducirse de forma controlada mediante herramientas como **CyberChef**, lo que permite obtener el siguiente estadio del payload, que denominaremos **stage2.js**.

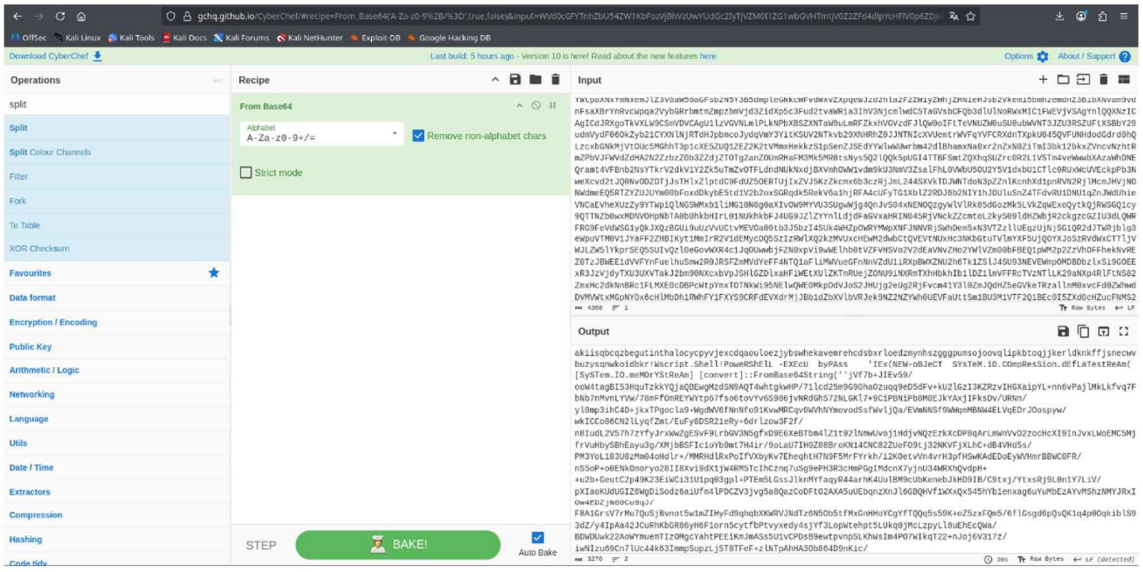


El fichero resultante contiene un número significativo de variables declaradas, pero nunca utilizadas, un indicio claro de ofuscación deliberada.

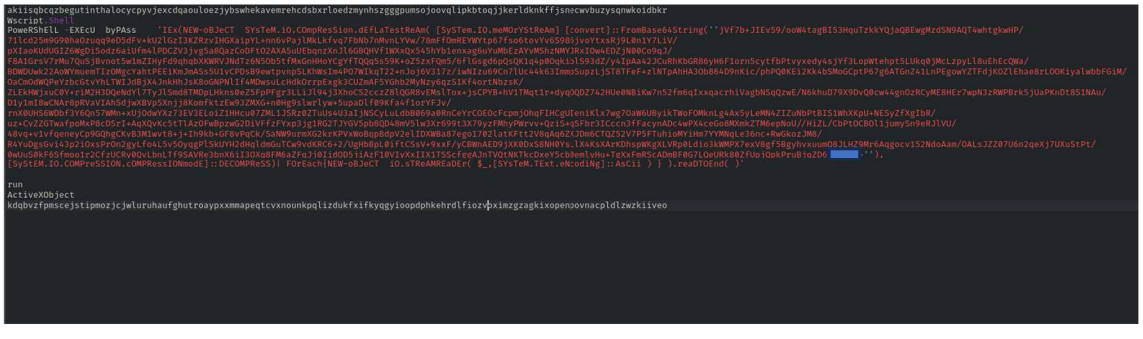
```
khhwtarcadepz = function(suadklsw) {
var twgoctfv={};
var qhypodermichff = 74142;
var vyafisoftcoverqa = false;
var aqMonsantoo = "undefined";
var qluypartitionswl = false;
var zwbwopamphletkzca = false;
var iibigrqb;
var kyieithero = null;
var xglisteniclnull;
var tsrjtemplelrhz = "undefined";
var mdljwdetachw = null;
var yghnplatypuszfwv = 60163;
var gqpsaltyoem = null;
var vbasemanozbcu = 39296;
var tsyphiliticddc = true;
var xulqdiurnald = "undefined";
var ldarlpqp=0;
```

Los primeros 149 líneas implementan una función de decodificación Base64 —designada como *xhhwtarcadepz*— cuyo propósito es revertir la codificación aplicada a un conjunto de seis cadenas que el propio script concatena para formar un único bloque Base64. Este bloque es posteriormente suministrado a la función de decodificación, dando lugar al siguiente componente de la cadena de infección.

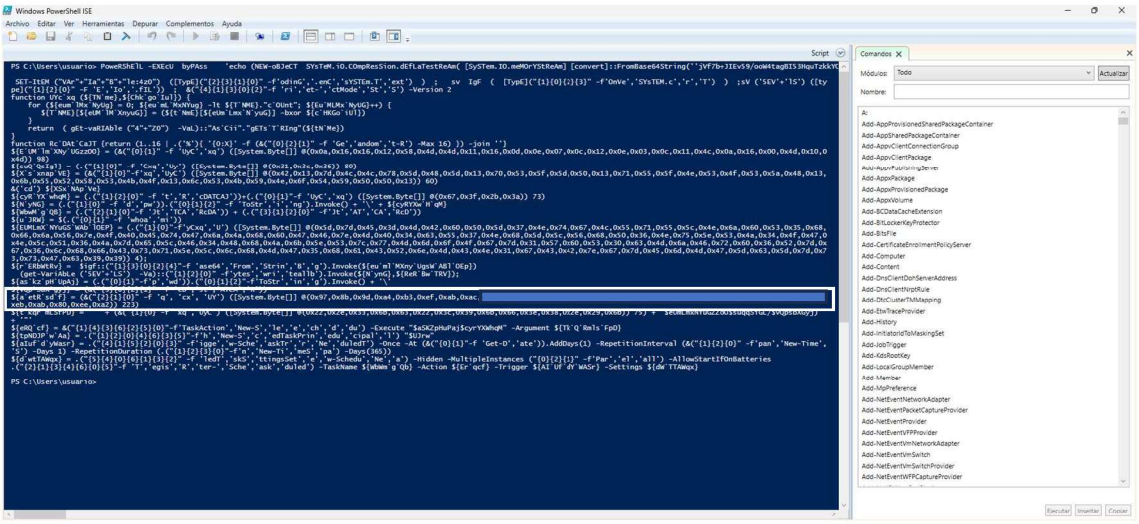
El resultado del proceso de decodificación previo es una cadena delimitada mediante el carácter **!**, que posteriormente se fragmenta para generar una lista de elementos que conforman el siguiente estadio del código malicioso.



Para facilitar la comprensión del flujo de ejecución, presento a continuación una representación visual del script resultante, donde se aprecia con claridad la estructura y las transformaciones aplicadas.



El código incluye una ventana emergente que informa al usuario de que el supuesto documento está dañado, seguida de una pausa artificial destinada a introducir latencia antes de ejecutar cualquier acción significativa. Superado este retardo, el script invoca la rutina de PowerShell observada en el fragmento anterior. Dicha rutina toma un *blob* codificado, lo decodifica desde Base64, lo descomprime mediante *deflate* y finalmente lo transfiere a **iex**, lo que provoca su ejecución directa en memoria.



El contenido decodificado constituye el siguiente estadio de la cadena de infección, que almacenamos como **stage3.ps1**.

Dentro de este script emerge una variable particularmente relevante. Aunque no participa en el flujo de ejecución, su contenido codificado puede revertirse para obtener la bandera final del desafío.