‘July Pay Requisition Approved.HTM’

Diego Alba

Malware Description

A malicious phishing site mimicking a microsoft sign-in page. Initial payload was delivered through email and included the above file: “July Pay Requisition Approved.HTM”

Static Analysis

The following steps were done for the initial static analysis:

(1) File Deobfuscation
(2) Link/String Extraction

The individual findings are reported in the subsections below.

(1) *File Deobfuscation*

An initial review of the file using the linux “cat” command revealed that the “.htm” file was only populated with a <script> tag.

```
<script>
     window.token
="JFJjl4jGAdem5Xwru2UsQ3Q2r";
     window.mail
="cody.banker.first@gmail.com"; //
throw-away email to maintain
privacy
     window.file
="https://uvah.tk/wp-content/er3.ph
p";
       window.ok = "";
     window.incr = "";
var _0x30637a=...
// more code below
```

After renaming all variables, I began to piece together how this malicious html file works.

There are 3 functions in the initial html file, which after deobfuscating, I’ve named appropriately: get_itm, get_arr, and an Immediately Invoked Executable Function (IIEF) which is at the heart of the program.

get_itm:

```
function get_itm(prm1) {
   return get_arr()[prm1 - 184];
}
```

An experience programmer might notice that this is a simple getter function which could simply be rewritten as a simple

```
return get_arr()[prm1]
```

But this is not possible due to the context it is used in! As shown below…

```
(function (prm1, prm2) {
   arr = prm1();
   var count = 0;
   while (true) {
     try {
       var wierd_int =
parseInt(get_itm(187)) / 1 + /*
more code */ ;
       if (wierd_int === prm2)
break; else arr.push(arr.shift());
     } catch (_0x43efba) {
       arr.push(arr.shift());
     }
   }
 }(get_arr, 389808),
document[get_itm(189)](unescape(get
_itm(196))));
```

The above obfuscated IIEF essentially calls the get_arr function and cycles through it like a circular queue until the value wierd_int is equivalent to 389808. My educated guess is that this is all done through an obfuscation engine. This can be rewritten as:

```
((e) => {

document["write"](unescape(html_inj
))})
```

where the variable html_inj is part of the array that is retrieved after each call to get_arr.

From our initial round of static analysis it is clear that this malicious javascript code injects html into the browser.

(2) *Link/String Extraction*
Now that the general background for the html file has been provided, the following links have been extracted from the file (note: for other malware reports, the list may be bigger!):

https://uvah.tk/wp-content/er3.php

- Doesn't seem to be reachable by simply typing it into the browser.

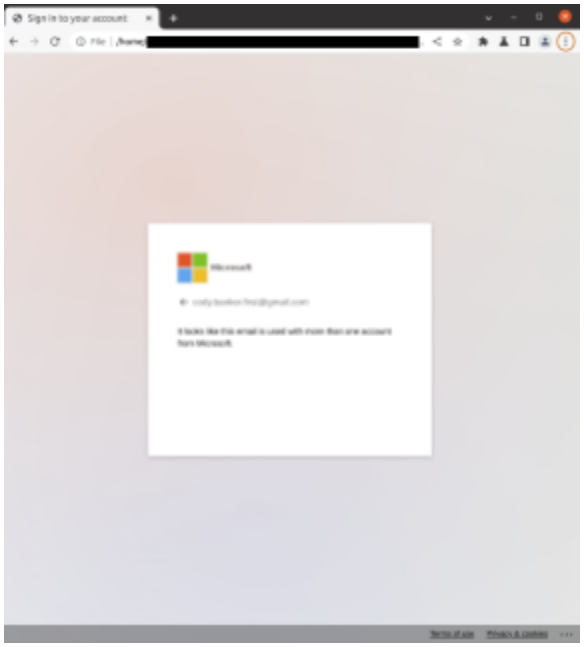
(3) *Additional Comments*
There are a series of variables, which were shown earlier, but are mentioned here as well.

```
window.token =
"JFJjl4jGAdem5Xwru2UsQ3Q2r";
 window.mail =
"cody.banker.first@gmail.com";
 window.file =
"https://uvah.tk/wp-content/er3.php
";
 window.ok = "";
 window.incr = "";
```

From the general comments found in the original file, it seems as if the malicious actor on the other end of this phishing campaign found this file and simply copied/pasted it.

Dynamic Analysis
I performed a series of dynamic analysis rounds using BurpSuite to extract information about the network traffic, as well as FireFox to perform some debugging regarding function execution.

Found below are some screenshots of the website (using BurpSuite CE).

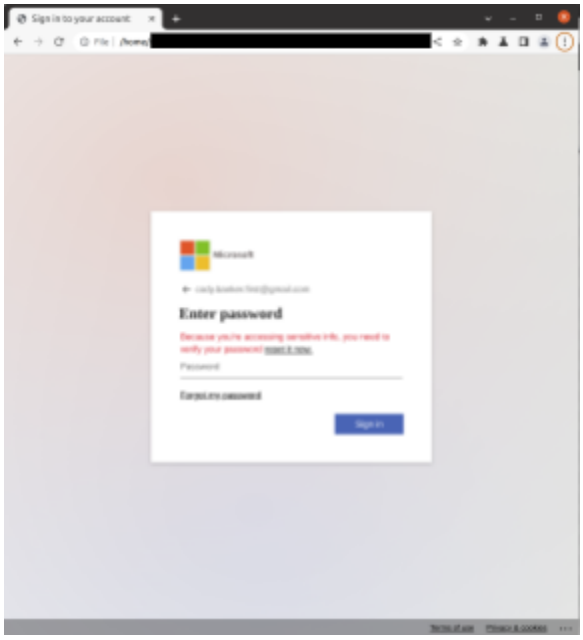


The above website queries **(code.jquery.com.de [38.34.165.183])** which has been flagged on virustotal by 1 vendor (Fortinet).

The user is then forwarded to: **(code.jquery.quest [104.21.93.234])**. The interesting thing about this is that the window.token variable from the picture on the left is sent as a parameter to a different php file.

The email parameter is also sent to another php file found at: **(code.jquery.quest [172.67.216.137])**.

Likewise the browser changes to the following:



This proves to be a fairly convincing microsoft login page!

Summary

MD5: 29b744631209678c0919e3da204d36f5
sha256: c6a981246ccf4a04df73b83374d0de2ff078e1e415a83bbbd66ff5665f46a53d

The above file injects malicious javascript and obfuscates its purpose in an attempt to throw off security analysts and possibly anti-virus software. The redirect chain in the dynamic analysis contained known malicious IPs (using AbuseIPDB). The injected javascript code will be included as a separate write-up.

In case there were any errors in my analysis, or for more information including but not limited to constructive criticism: contact me.