Flerken v1.0 Specification Document

Yao Zhang and Zhiyang Zeng
Tencent Blade Team

April 26, 2019

Abstract

Flerken is an open-source obfuscated command line detection tool that works for both Windows (CMD and Powershell) and Linux (Bash) commands. To the best of our knowledge, **Flerken is the first tool that support cross-platform obfuscation detection feature**. In this specification document, we share an overview of how Flerken works and how simply Flerken can be leveraged. The effectiveness of Flerken is evaluated via representative black/white command samples. We also compare our approach with highly-related schemes such as Revoke-Obfuscation. Please feel free to visit our beta version website [1] and Flerken's Github page [2].

1 Introduction

Command line obfuscation has been proved to be a non-negligible factor in fileless malware or malicious actors that are "living off the land". To bypass signature-based detection, dedicated obfuscation techniques are shown to be used by red-team penetrations [3, 4] and even APT activities [5]. Meanwhile, numerous obfuscators (namely tools perform syntax transformation) are open sourced [6, 7, 8, 9], thus making obfuscating given commands increasingly effortless.

However, the number of suitable defenses remains to be few. For Linux command line obfuscation, we can barely find any detection tools. Concerning defenses against Windows command obfuscation, existing schemes turn out to either lack of toolization [10], or only partially resolve the entire problem [11, 12], sometimes even inaccurately.

To better facilitate obfuscation detection, we have proposed Flerken, a toolized platform that can be used to detect both Windows (CMD and Powershell) and Linux (Bash) commands. The name of Flerken is inspired by a cat-like yet extremely powerful creature from Marvel world. Flerken is build on the basis of carefully collection of black/white samples, and can be divided into two sub-schemes, namely Kindle (Windows obfuscation detector) and Octopus (Linux obfuscation detector). To help optimize Flerken's classification performance, we adopt techniques such as machine learning, bi-directional feature filter ring, script sandboxing (more details described in Section 2). Hereby, we highlight the functional properties Flerken basically satisfies as follows:

- **Scalability.** Flerken supports cross-platform obfuscation detection. Namely, both Windows and Linux command line stings can be queried to check if there is any obfuscation.
- **Platform Recognition.** When you use Flerken, ideally you select which platform (Windows/Linux) the tested command comes from. However, if you lack of such information, you can just query with a default option and Flerken will return a classification result with the likely platform information.
- **Accuracy.** Despite plenty of optimization work still needed, Flerken is adequate to distinguish most Windows/Linux command obfuscations. In fact, Flerken has been adopted by Tencent in many security investigations of server endpoints (with a scale on the order of millions).

- **Easy of Use.** Flerken v1.0 now is accessible through its official webpage [1]. All you have to do is to paste into the command sting that you want to analyze. No specific input file format like Revoke-Obfuscation [11] is required. We have also open-sourced Flerken on Github [2] so you can build your own detector on demand. Besides, we will soon provide an API of Flerken to make the usage more conveniently.

Note that unlike other de-obfuscation tool that brings certain command or script back to plaintext format [13], Flerken focuses on distinguishing obfuscation from those non-obfuscated commands in terms of syntax. Semantic analysis (determining whether the command is benign of malicious) is also out of our scope.

2 Design

This section gives an overview of how Flerken works. The entire approach is composed of two detectors, to analyze Windows and Linux command obfuscation, respectively.

2.1 Windows Obfuscation

Kindle is a sub-scheme of Flerken in terms of CMD and Powershell obfuscated command detection. Our insight here is, just like humans usually do, readability can serve as a key feature in distinguishing obfuscated commands from the others. Therefore, when performing feature engineering based on readability, it would be much easier to differentiate those unreadable obfuscated texts.

In Kindle, we consider two types of composite features, namely the ratio of **readable words** and the ratio of **readable characters**. Specifically, readable word is defined with consideration of the following aspects:

- Whether the word is composed of English alphabets.
- Ratio interval of vowel letters.
- Consecutive occurrences of the same letter.
- Ratio interval of uppercase letters.
- Whether the first letter is capitalized.
- Word length.

Under such definition, we can extract the entire input string and check all included words. In order to correctly label the words, a whitelist of common-used words (e.g., abbreviations) that might be unreadable can be leveraged.

Considering readable characters, we define those characters (e.g., '(', ')', '=', '.') that are often used in non-obfuscated commands. And special characters that are barely appeared in a terminal operation are categorized as unreadable characters, such as '^','&', '%', '@'.

In addition to the readability-related features, Kindle also takes a few statistical features into consideration. These features include: (1) *the ratio of characters*; (2) *the ratio of spaces in the command*; (3) *the length of the input command.* Combining all features above, we deploy decision tree machine learning technique to distinguish obfuscated and non-obfuscated commands. The training data (positive samples) used here is mainly contributed by Daniel Bohannon's Invoke-DOSfuscation [6] and Invoke-Obfuscation [7]. We also collected millions of commands (negative samples) from a testbed server cluster of Tencent.

Note that some coding-involved obfuscation methods (e.g., base64) may not be precisely covered by the readability-based detection described above. We have implemented another decision tree for better awareness of these obfuscations. The process is essentially similar, and we omit it here for simplicity.

2.2 Linux Obfuscation

The other sub-scheme of Flerken is Octopus, which takes care of Linux Bash obfuscation. Some great positive samples are summarized on Malwrologist's Twitter page [3]. Yet the sample scale is still quite limited compared to the number of negative samples we collected from the server testbed. Due to this imbalance, we rebuild a different scheme other than Kindle.

Based on the obfuscated samples, we have concluded around 15 different types of syntax skills and tried to generalize their features accordingly. These feature types including (1) *Variable Name Bypass*, (2) *Substitution Syntax*, (3) *Offset Control*, (4) *Hex or Unicode syntax*, etc. Different generalizing degrees will influence the size of the associated output set. Such skill representations form a so-called grey **feature filter ring** that will effectively label suspicious commands, even though the command inputs may be enormous.

Each filter type will be assigned with a corresponding whitelist type that contains false positives obtained from negative samples. All whitelist types will form a white feature filter ring. Using such bi-directional feature rings will help better confirm whether the input is obfuscated or not.

To enable better analyzing performance, a script sandboxing technique can be further used to exclude false positives. Specifically, if a command still remains to be positive after bi-direction feature filter rings, a 'sh -x' **script sandbox** will be leveraged to obtain running status of the command. Hence, methods such as similarity estimation or signature-based matching can be adopted to finalize our analysis.

Note that such obfuscation skills that slightly change a plaintext command are not yet in Flerken's scope (e.g., obfuscation that only use backticks, backslash, or single/double quotation marks). We leave this as future work.

3 How to Use Flerken

This section gives a guidance on how to use Flerken through our webpage [1]. If you are aimed at building a private Flerken, please check our Github page [2] where you can find more details on installation and environment requirements, etc.

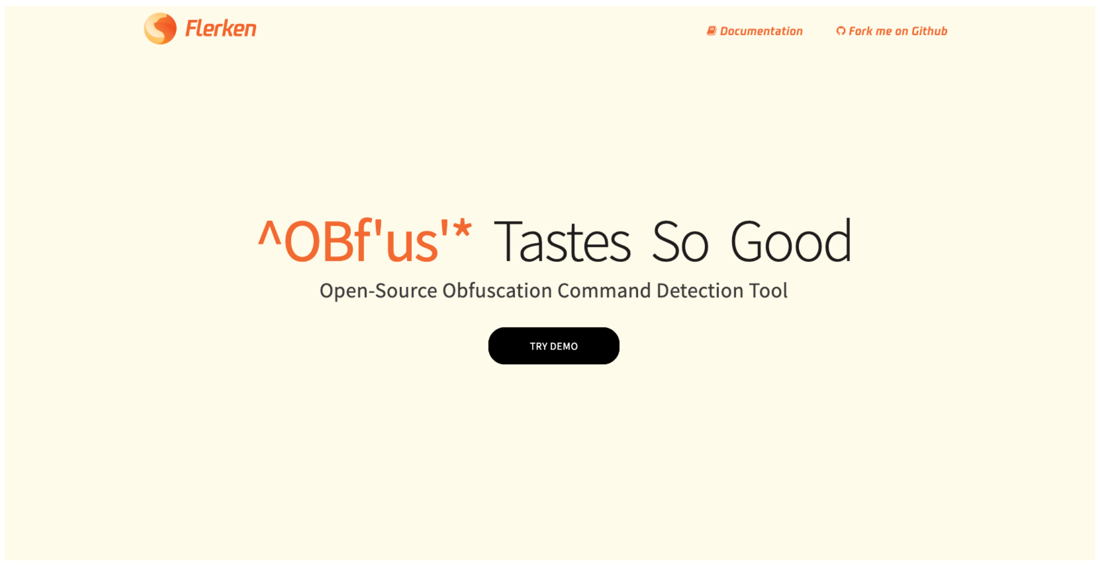
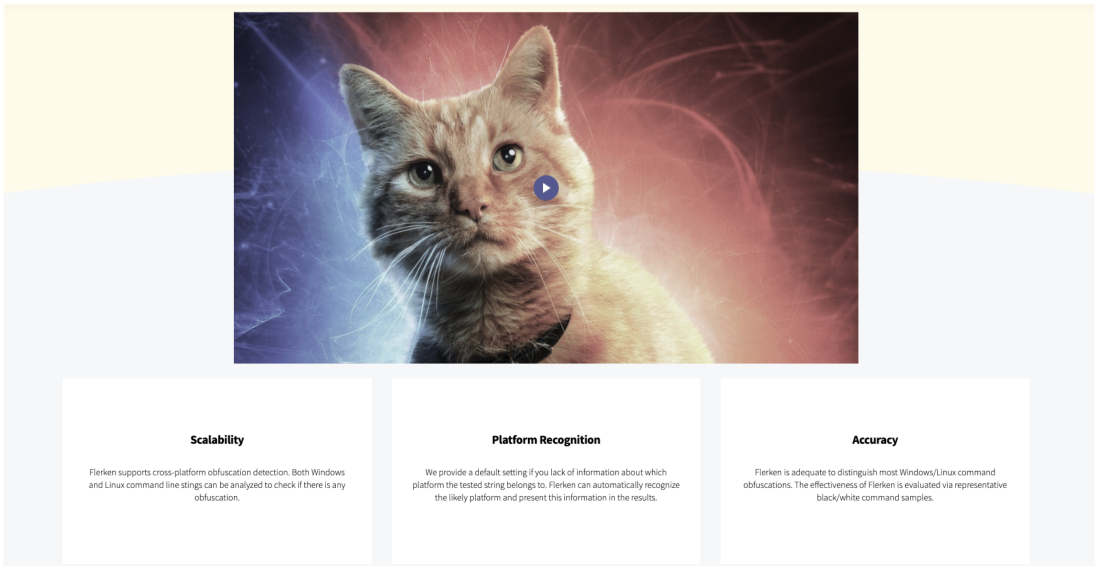


Figure 1: Flerken landing page (I).

Figure 1 and Figure 2 show Flerken's landing page, where you can find links to Flerken specification documentation and Github page. We also added a "How To Use" video with highlighted properties (as shown in Figure 2). Click "TRY DEMO" button will lead you to the detection page (Figure 3).



Scalability

Flerken supports cross-platform obfuscation detection. Both Windows and Linux command line stings can be analyzed to check if there is any obfuscation.

Platform Recognition

We provide a default setting if you lack of information about which platform the tested string belongs to. Flerken can automatically recognize the likely platform and present this information in the results.

Accuracy

Flerken is adequate to distinguish most Windows/Linux command obfuscations. The effectiveness of Flerken is evaluated via representative black/white command samples.

Figure 2: Flerken landing page (II).

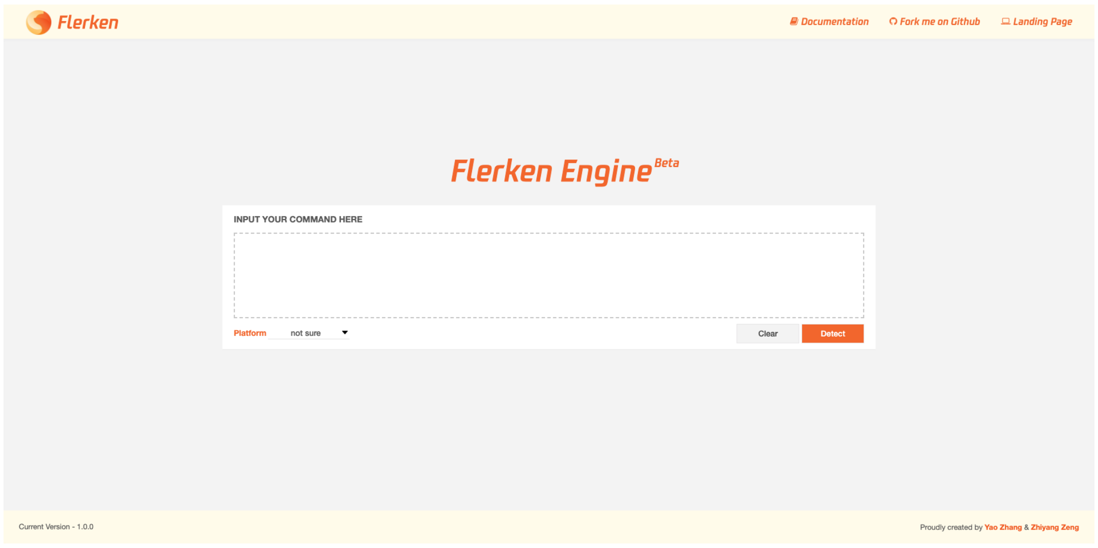
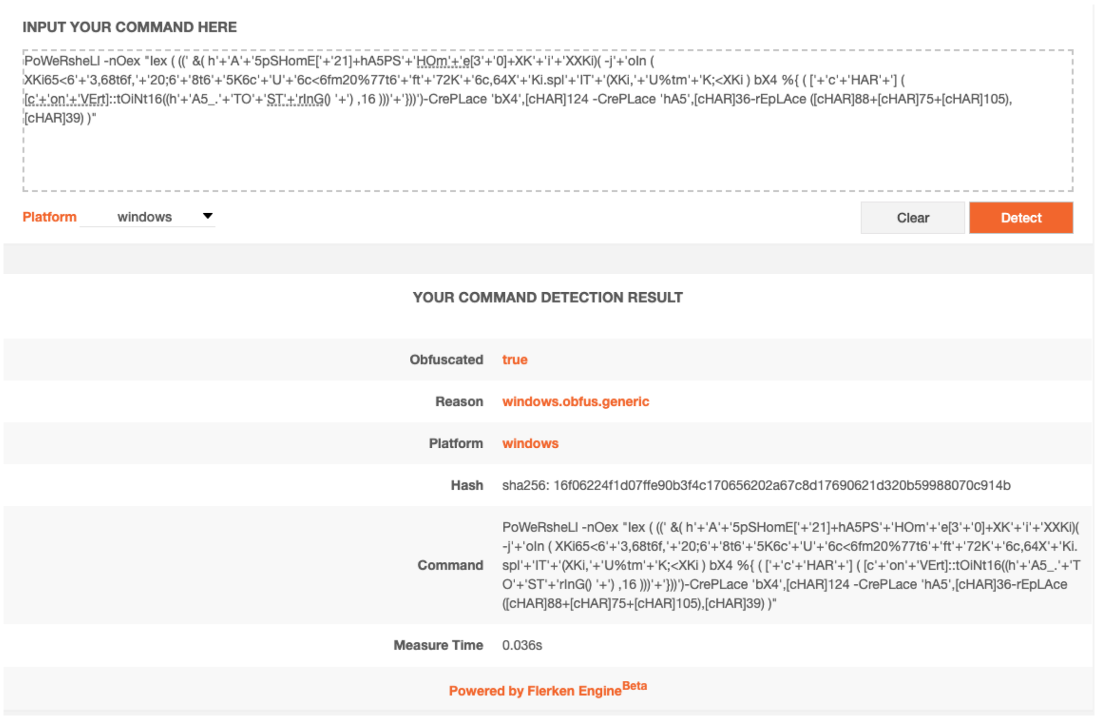


Figure 3: Flerken detection page.

In the detection page, you simply put your command into the text box. You are suggested to select the corresponding platform (at the small platform box below) to get a more precise detection. If you are uncertain about what kind of obfuscation you are querying, just go with a default setting "not sure". Flerken can automatically recognize the likely platform that the command belongs to and present the information in the results. When you need to re-enter any commands, just click "clear" button.

Now let's have a look at the result page. Figure 4, Figure 5, and Figure 6 show the results of different command analyses, respectively. As we can see from the figures, the result field basically contains the following contents:

- **Obfuscated**: Gives judgement on obfuscation. There are three different types: *True*, *False*, and *Suspicious*.
- **Reason**: Shows which type of obfuscation the command matches in general. Now a couple of labels are used, such as *Linux.obfus.generic*, *Linux.obfus.special*, *Linux.obfus.graphic*, *Win-*


INPUT YOUR COMMAND HERE

```
PoWeRsheLl -nOex "Iex ( ((' &( h'+'A'+'5pSHomE['+'21]+hA5PS'+'HOm'+'e[3'+'0]+XK'+'i'+'XXKi)( -j'+'oIn (
XKi65<6'+'3,68t6f,'+'20;6'+'8t6'+'5K6c'+'U'+'6c<6fm20%77t6'+'ft'+'72K'+'6c,64X'+'Ki.spl'+'IT'+'(XKi,'+'U%tm'+'K;<XKi ) bX4 %{ ( ['+'c'+'HAR'+'] (
[c'+'on'+'VErt]::tOiNt16((h'+'A5_.'+'TO'+'ST'+'rInG() '+') ,16 )))'+'}))')-CrePLace 'bX4',[cHAR]124 -CrePLace 'hA5',[cHAR]36-rEpLAce ([cHAR]88+[cHAR]75+[cHAR]105),
[cHAR]39) )"
```

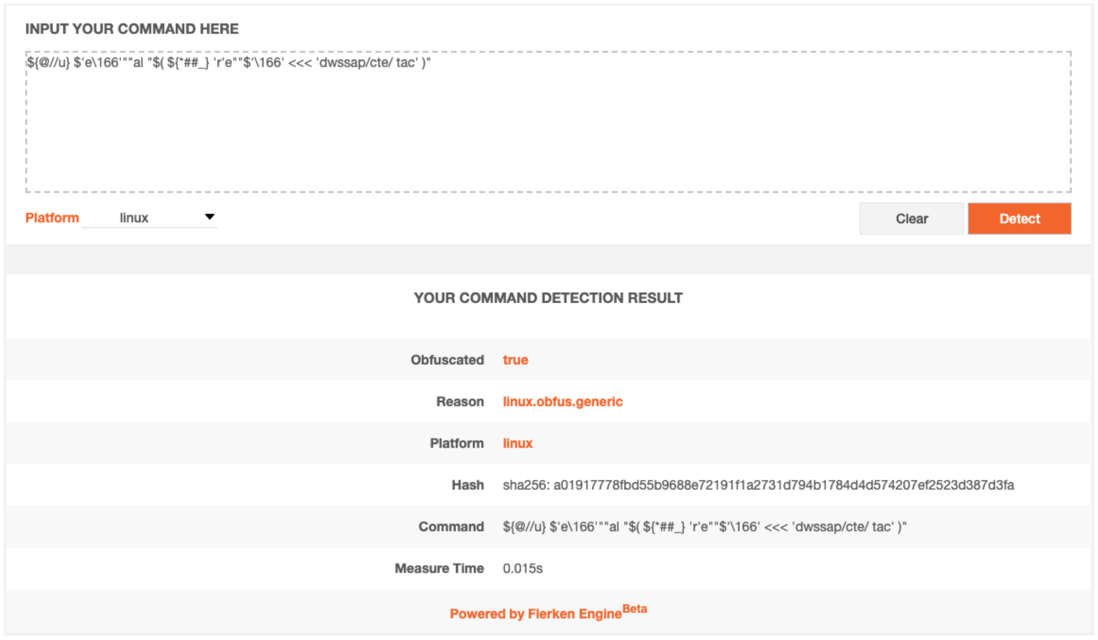
Platform windows

YOUR COMMAND DETECTION RESULT

Obfuscated	true
Reason	windows.obfus.generic
Platform	windows
Hash	sha256: 16f06224f1d07ffe90b3f4c170656202a67c8d17690621d320b59988070c914b
Command	PoWeRsheLl -nOex "Iex (((' &(h'+'A'+'5pSHomE['+'21]+hA5PS'+'HOm'+'e[3'+'0]+XK'+'i'+'XXKi)(-j'+'oIn (XKi65<6'+'3,68t6f,'+'20;6'+'8t6'+'5K6c'+'U'+'6c<6fm20%77t6'+'ft'+'72K'+'6c,64X'+'Ki.spl'+'IT'+'(XKi,'+'U%tm'+'K;<XKi) bX4 %{ (['+'c'+'HAR'+'] ([c'+'on'+'VErt]::tOiNt16((h'+'A5_.'+'TO'+'ST'+'rInG() '+') ,16)))'+'}))')-CrePLace 'bX4',[cHAR]124 -CrePLace 'hA5',[cHAR]36-rEpLAce ([cHAR]88+[cHAR]75+[cHAR]105),[cHAR]39))"
Measure Time	0.036s

Powered by Flerken Engine^Beta

Figure 4: Analyzing result of a given Windows command (obfuscation detected).


INPUT YOUR COMMAND HERE

```
${@//u} $'e\166'""al "$( ${*##_} 'r'e""$'\166' <<< 'dwssap/cte/ tac' )"
```

Platform linux

YOUR COMMAND DETECTION RESULT

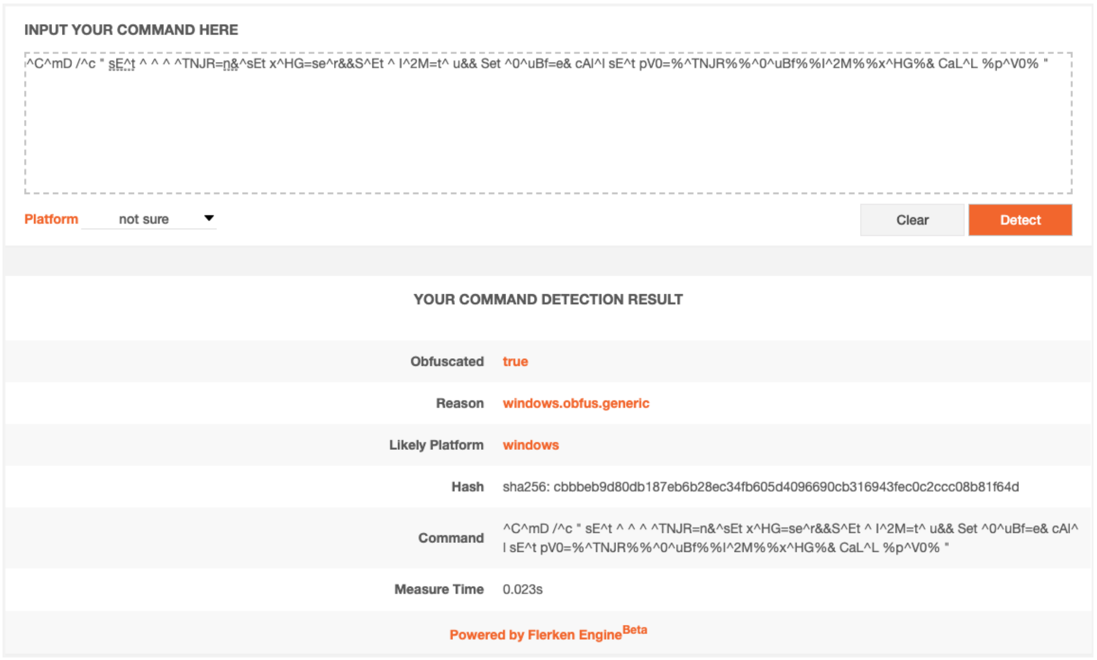
Obfuscated	true
Reason	linux.obfus.generic
Platform	linux
Hash	sha256: a01917778fbd55b9688e72191f1a2731d794b1784d4d574207ef2523d387d3fa
Command	${@//u} $'e\166'""al "$(${*##_} 'r'e""$'\166' <<< 'dwssap/cte/ tac')"
Measure Time	0.015s

Powered by Flerken Engine^Beta

Figure 5: Analyzing result of a given Linux command (obfuscation detected).

dows.obfus.generic, *Windows.obfus.special*, *Windows.suspicious.generic*.

- **Platform**: Platform information. If selected before detection, the result will just reflect this information. Otherwise, a platform estimation will be given by the Flerken engine.
- **Hash**: SHA-256 hash of the command input.
- **Command**: Shows the input command, only first 1000 bytes visible.


INPUT YOUR COMMAND HERE

```
^C^mD /^c " sE^t ^ ^ ^ ^TNJR=n&^sEt x^HG=se^r&&S^Et ^ I^2M=t^ u&& Set ^0^uBf=e& cAl^I sE^t pV0=%^TNJR%%^0^uBf%%I^2M%%x^HG%& CaL^L %p^V0% "
```

Platform: not sure

YOUR COMMAND DETECTION RESULT

Obfuscated	true
Reason	windows.obfus.generic
Likely Platform	windows
Hash	sha256: cbbbeb9d80db187eb6b28ec34fb605d4096690cb316943fec0c2ccc08b81f64d
Command	^C^mD /^c " sE^t ^ ^ ^ ^TNJR=n&^sEt x^HG=se^r&&S^Et ^ I^2M=t^ u&& Set ^0^uBf=e& cAl^I sE^t pV0=%^TNJR%%^0^uBf%%I^2M%%x^HG%& CaL^L %p^V0% "
Measure Time	0.023s

Figure 6: Analyzing result of a given command with platform unspecified (obfuscation detected).

- **Measure Time**: Total time used.

4 Evaluation

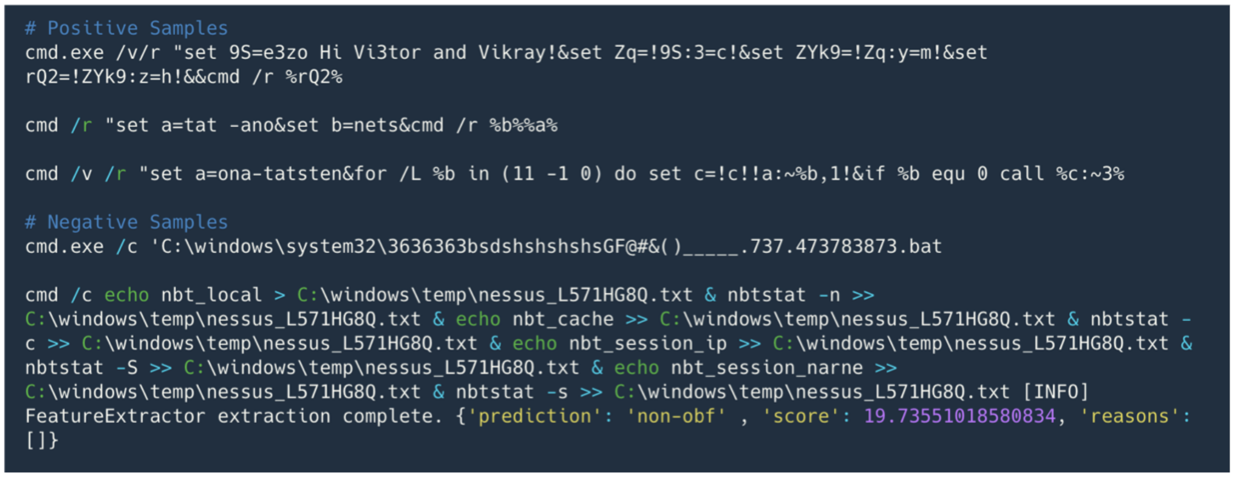
4.1 Defense on Windows Obfuscation

Methodology Analysis. First, we compare our scheme with related detection proposals [10, 11, 12]. Schemes [11, 12] are both intended to distinguish Powershell command line obfuscation, thus only partially solving Windows Obfuscation issue. Specifically, Revoke-Obfuscation extracts features based on Powershell Abstract Syntax Tree (AST). A feature vector with nearly 5000 dimensions is thus established. In contrast, Flerken selects features with a much scalable size, and could be more interpretable with those readability-based definitions. Moreover, Flerken also supports CMD command line detection.

A more recent proposal [10] by FireEye is adequate to detect both CMD and Powershell command line obfuscations. Unfortunately, no open-source tool has been released until now. In comparison, Flerken's Kindle sub-scheme also performs near perfect with metrics (such as F1-score, precision, and recall) all being close to 1.0, which is comparable to the Gradient Boosted Tree (GBT) model mentioned in [10].

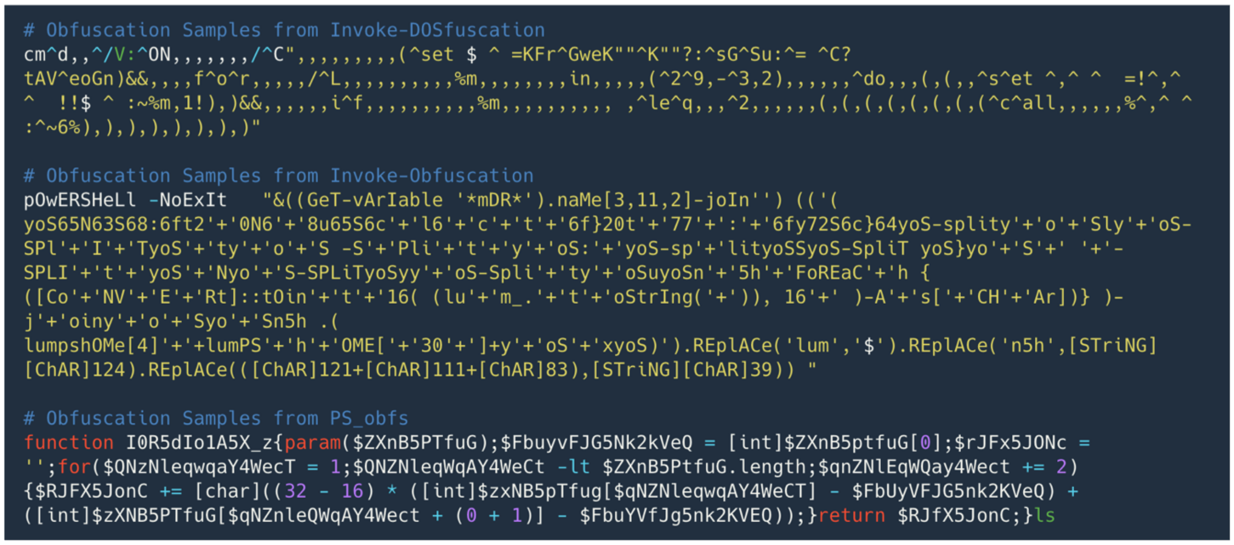
Sample tests. We first leverage black and white samples in [10] and test them against Flerken. As shown in Figure 7, the first three commands are obfuscated texts (provided by Daniel Bohannon) that are usually difficult to detect, and the last two texts are actually non-obfuscated Windows command lines. As a result, Flerken can successfully identify all black/white samples above.

We further evaluate Flerken with samples generated using three obfuscators, namely Invoke-DOSfuscation (CMD Obfuscation Generator) [6], Invoke-Obfuscation (Powershell obfuscator) [7], and PS_obfs (Powershell obfuscation engine) [14]. For each obfuscator, we randomly generated 1000 samples and use them as Flerken inputs (some of which are shown in Figure 8. **Consequently, Flerken can distinguish most all of the tested samples** (nearly 100% detection rate).



```
# Positive Samples
cmd.exe /v/r "set 9S=e3zo Hi Vi3tor and Vikray!&set Zq=!9S:3=c!&set ZYk9=!Zq:y=m!&set
rQ2=!ZYk9:z=h!&&cmd /r %rQ2%

cmd /r "set a=tat -ano&set b=nets&cmd /r %b%%a%

cmd /v /r "set a=ona-tatsten&for /L %b in (11 -1 0) do set c=!c!!a:~%b,1!&if %b equ 0 call %c:~3%

# Negative Samples
cmd.exe /c 'C:\windows\system32\3636363bsdshshshshsGF@#&()_____.737.473783873.bat

cmd /c echo nbt_local > C:\windows\temp\nessus_L571HG8Q.txt & nbtstat -n >>
C:\windows\temp\nessus_L571HG8Q.txt & echo nbt_cache >> C:\windows\temp\nessus_L571HG8Q.txt & nbtstat -
c >> C:\windows\temp\nessus_L571HG8Q.txt & echo nbt_session_ip >> C:\windows\temp\nessus_L571HG8Q.txt &
nbtstat -S >> C:\windows\temp\nessus_L571HG8Q.txt & echo nbt_session_narne >>
C:\windows\temp\nessus_L571HG8Q.txt & nbtstat -s >> C:\windows\temp\nessus_L571HG8Q.txt [INFO]
FeatureExtractor extraction complete. {'prediction': 'non-obf' , 'score': 19.73551018580834, 'reasons':
[]}
```

Figure 7: Black & white command samples according to FireEye's research blog.



```
# Obfuscation Samples from Invoke-DOSfuscation
cm^d,,^/V:^ON,,,,,,,,/^C",,,,,,,,,,(^set $ ^ =KFr^GweK""^K""?:^sG^Su:^= ^C?
tAV^eoGn)&&,,,,f^o^r,,,,,,/^L,,,,,,,,,,,%m,,,,,,,,,in,,,,,(^2^9,-^3,2),,,,,,^do,,,(,(,,^s^et ^,^ ^  =!^,^
^  !!$ ^ :~%m,1!),)&&,,,,,,i^f,,,,,,,,,,,%m,,,,,,,,,, ,^le^q,,,^2,,,,,,(,(,(,(,(,(,(,(^c^all,,,,,,%^,^ ^
:^~6%),),),),),),),)"

# Obfuscation Samples from Invoke-Obfuscation
pOwERSHeLl -NoExIt   "&((GeT-vArIable '*mDR*').naMe[3,11,2]-joIn'') (('(
yoS65N63S68:6ft2'+'0N6'+'8u65S6c'+'l6'+'c'+'t'+'6f}20t'+'77'+':'+'6fy72S6c}64yoS-splity'+'o'+'Sly'+'oS-
SPl'+'I'+'TyoS'+'ty'+'o'+'S -S'+'Pli'+'t'+'y'+'oS:'+'yoS-sp'+'lityoSSyoS-SpliT yoS}yo'+'S'+' '+'-
SPLI'+'t'+'yoS'+'Nyo'+'S-SPLiTyoSyy'+'oS-Spli'+'ty'+'oSuyoSn'+'5h'+'FoREaC'+'h {
([Co'+'NV'+'E'+'Rt]::tOin'+'t'+'16( (lu'+'m_.'+'t'+'oStrIng('+')), 16'+' )-A'+'s['+'CH'+'Ar])} )-
j'+'oiny'+'o'+'Syo'+'Sn5h .(
lumpshOMe[4]'+'+lumPS'+'h'+'OME['+'30'+']+y'+'oS'+'xyoS)').REplACe('lum','$').REplACe('n5h',[STriNG]
[ChAR]124).REplACe(([ChAR]121+[ChAR]111+[ChAR]83),[STriNG][ChAR]39)) "

# Obfuscation Samples from PS_obfs
function I0R5dIo1A5X_z{param($ZXnB5PTfuG);$FbuyvFJG5Nk2kVeQ = [int]$ZXnB5ptfuG[0];$rJFx5JONc =
'';for($QNzNleqwqaY4WecT = 1;$QNZNleqWqAY4WeCt -lt $ZXnB5PtfuG.length;$qnZNlEqWQay4Wect += 2)
{$RJFX5JonC += [char]((32 - 16) * ([int]$zxNB5pTfug[$qNZNleqwqAY4WeCT] - $FbUyVFJG5nk2KVeQ) +
([int]$zXNB5PTfuG[$qNZnleQWqAY4Wect + (0 + 1)] - $FbuYVfJg5nk2KVEQ));}return $RJfX5JonC;}ls
```

Figure 8: Some obtained samples from Invoke-DOSfuscation, Invoke-Obfuscation, and PS_obfs.

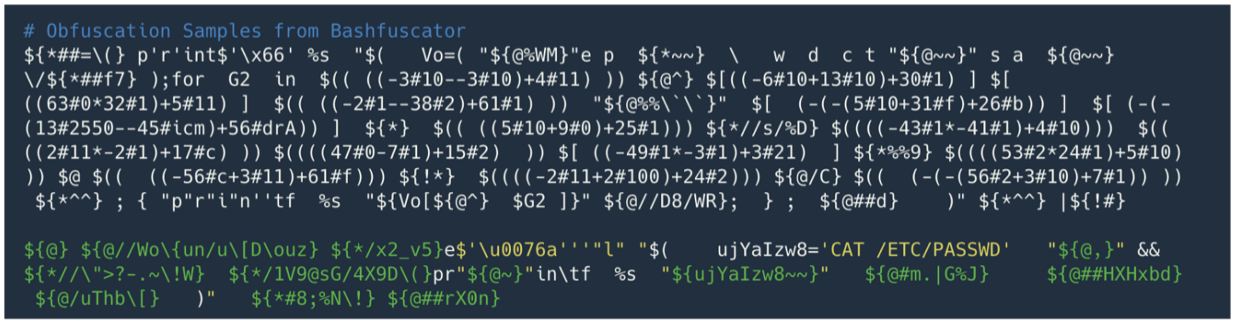
4.2 Defense on Linux Obfuscation

Now we evaluate Flerken against some Linux command obfuscations. Bashfuscator is a perfect Bash obfuscation framework that released recently. We hereby deploy this tool and randomly generate 1000 Bash commands. Some of the commands are shown in Figure 9. **After testing all these samples, we find that our identification rate is more than 99%**.

Note that Flerken also covers most of the obfuscation techniques introduced in [3, 4]. Yet as we mentioned in Section 2, currently defense skills against slight syntax changing (of a given command) are not fully supported by Flerken (e.g., obfuscation that only use backticks, backslash, or single/double quotation marks). We leave this as future work.

5 Conclusion

This document gives an overview of Flerken, a command-line obfuscation detection tool. We have illustrated how to use Flerken and confirmed the advantages of Flerken in identifying cross-platform



```
# Obfuscation Samples from Bashfuscator
${*##=\(} p'r'int$'\x66' %s  "$(   Vo=( "${@%WM}"e p  ${*~~}  \   w  d  c t "${@~~}" s a  ${@~~}
\/${*##f7} );for  G2  in  $(( ((-3#10--3#10)+4#11) )) ${@^} $[((-6#10+13#10)+30#1) ] $[
((63#0*32#1)+5#11) ]  $(( ((-2#1--38#2)+61#1) ))  "${@%%\`\`}"  $[  (-(-(5#10+31#f)+26#b)) ]  $[ (-(-
(13#2550--45#icm)+56#drA)) ]  ${*}  $(( ((5#10+9#0)+25#1))) ${*//s/%D} $((((-43#1*-41#1)+4#10)))  $((
((2#11*-2#1)+17#c) )) $((((47#0-7#1)+15#2)  )) $[ ((-49#1*-3#1)+3#21)  ] ${*%%9} $((((53#2*24#1)+5#10)
)) $@ $((  ((-56#c+3#11)+61#f))) ${!*}  $((((-2#11+2#100)+24#2))) ${@/C} $((  (-(-(56#2+3#10)+7#1)) ))
 ${*^^} ; { "p"r"i"n''tf  %s  "${Vo[${@^}  $G2 ]}" ${@//D8/WR};  } ;  ${@##d}    )" ${*^^} |${!#}

${@} ${@//Wo\{un/u\[D\ouz} ${*/x2_v5}e$'\u0076a'''"l" "$(    ujYaIzw8='CAT /ETC/PASSWD'   "${@,}" &&
${*//\">?-.~\!W}  ${*/1V9@sG/4X9D\(}pr"${@~}"in\tf  %s  "${ujYaIzw8~~}"   ${@#m.|G%J}     ${@##HXHxbd}
 ${@/uThb\[}   )"   ${*#8;%N\!} ${@##rX0n}
```

Figure 9: Some obtained samples from Bashfuscator.

command obfuscations.

We will soon release Flerken's APIs. Currently we are also working on some advanced functions such as script sandbox as well as detection plugins for slight plaintext obfuscations. Flerken is now available on our webpage [1] and its Github page [2]. We appreciate it very much for your feedbacks.

6 Acknowledgments

We would like to thank Bghost Zhu, Junbo Li, Haizhang Du, Conan Hu, and other colleagues of Tencent Blade Team for their insightful feedbacks throughout the project. We also thank Ning Liu, Junjun Luo, and Lake Hu for their valuable comments and support.

References

[1] Flerken: A Cross-Platform Obfuscation Command Detection Tool. `https://flerken.pro`.

[2] Flerken Github Project. `https://github.com/We5ter/Flerken`.

[3] Malwrologist's Twitter on #Linux #Bash #Obfuscation. `https://twitter.com/DissectMalware/status/1023682809368653826`.

[4] 101 Pen Test and Bug Bounty Tips & Tricks #1 - Bypassing Command Filters. `https://nfsec.pl/pentest/6135`.

[5] Obfuscation in the Wild: Targeted Attackers Lead the Way in Evasion Techniques. `https://www.fireeye.com/blog/threat-research/2017/06/obfuscation-in-the-wild.html`.

[6] Invoke-DOSfuscation: A PowerShell v2.0+ Compatible CMD.EXE Command Obfuscation Framework. `https://github.com/danielbohannon/Invoke-DOSfuscation`.

[7] Invoke-Obfuscation: A PowerShell v2.0+ Compatible PowerShell Command and Script Obfuscator. `https://github.com/danielbohannon/Invoke-Obfuscation`.

[8] A Fully Configurable and Extendable Bash Obfuscation Framework. `https://github.com/Bashfuscator/Bashfuscator`.

[9] Blind Bash: Obfuscate Your Bash Code. `https://github.com/Rizer0/Blind-Bash`.

[10] Obfuscated Command Line Detection Using Machine Learning. `https://www.fireeye.com/blog/threat-research/2018/11/obfuscated-command-line-detection-using-machine-learning.html`.

[11] Revoke-Obfuscation: A PowerShell v3.0+ Compatible PowerShell Obfuscation Detection Framework. `https://github.com/danielbohannon/Revoke-Obfuscation`.

[12] Obfuscated Powershell Detection with Markov Chains. `https://github.com/frknozr/detect-obfuscation`.

[13] FireEye Labs Obfuscated String Solver. `https://github.com/fireeye/flare-floss`.

[14] Microsoft Powershell Obfuscation Engine. `https://github.com/Mil4n0/PS_obfs`.