UNIVERSITY OF AMSTERDAM

Detecting Fileless Malicious Behaviour of .NET C2 Agents using ETW

Supervisors:
Leandro Velasco
Joao de Novais Marques

Course:
Research Project 1

Authors:
Alexander Bode
Niels Warnars

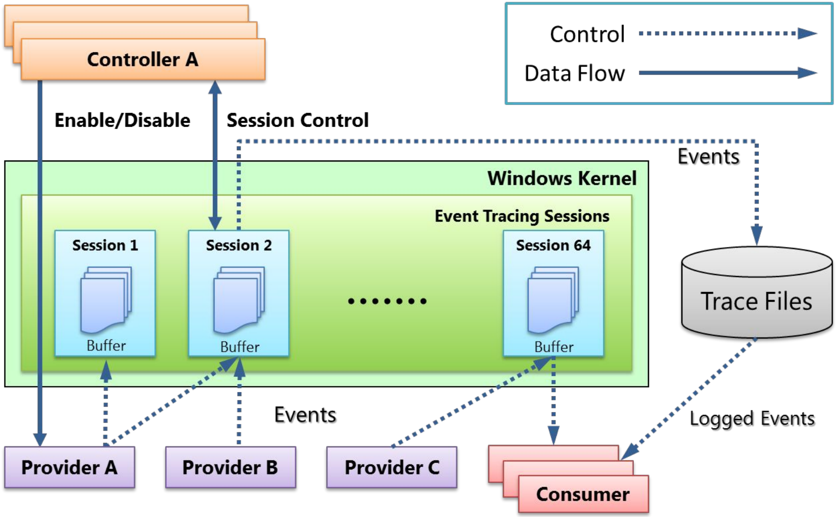
Event Tracing for Windows

Enables logging kernel or application data, since Windows 2000

Components of ETW

- Providers
- Controllers
- Consumers

ETW Architecture

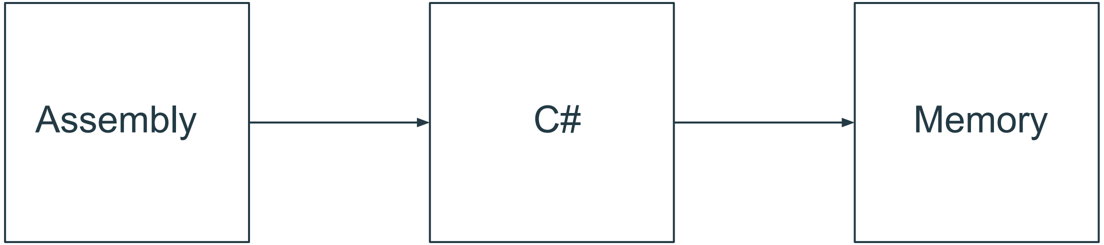


Source: Microsoft Docs, 2020

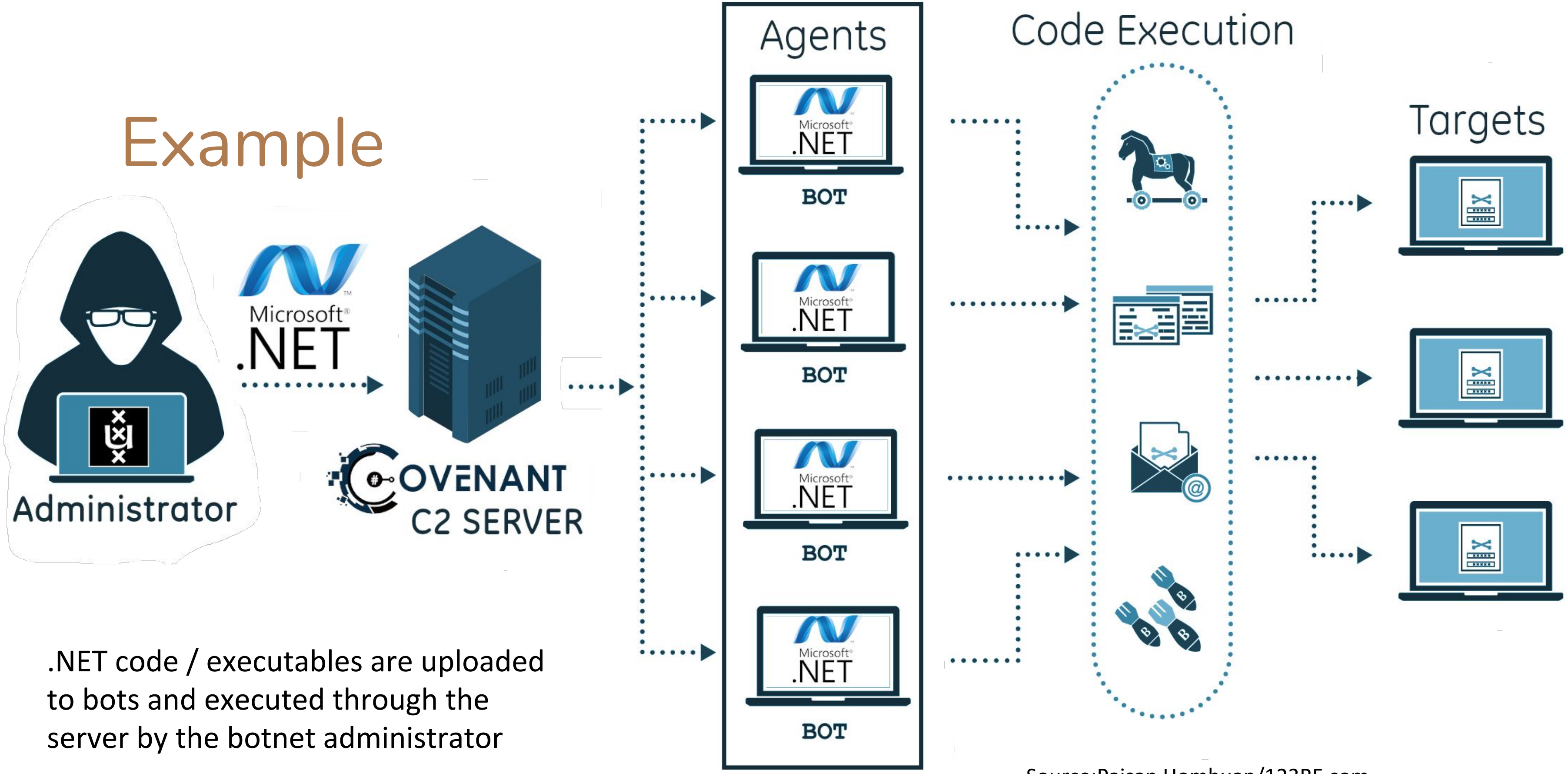
Fileless Malicious Behaviour of .NET C2 Agents

.NET assemblies can be dynamically loaded and executed into memory

- Using methods from the System.Reflection namespace
- Allowing remote execution of malicious code



Example



.NET code / executables are uploaded to bots and executed through the server by the botnet administrator

Source:Paisan Homhuan/123RF.com

Research Questions

Main Research Question

How can ETW be leveraged to detect fileless malicious behaviour of .NET agents used by popular C2 frameworks?

Sub Questions

What language-specific features can be used by .NET C2 agents for fileless attacks?

Which event types are relevant for detecting malicious .NET behaviour?

Importance

- Attackers shifting away from PowerShell to malicious .NET
- Logging and tracing support since Windows 2000
- Complexity and volume of data produced by ETW

Research Goals

- Find ways to detect .NET agents used by popular C2 frameworks using ETW
- Reduce false-positives and data volume
- Identify limitations of proposed detection methods

Current Research

Detection using ETW

- .NET code injection (F-Secure)
- Ransomware (CyberPoint)

Bypassing ETW

- For specific events, e.g., Asynchronous Procedure Calls (Tsukerman)
- Disable or delete ETW components (Palentir)
- ETW logs being renamed in the wild (Kaspersky)

Shortcomings

Detection using ETW

- Methods for detecting .NET code injection using ETW (F-Secure)
  - Inefficient research POC which uses the PyWintrace library
  - Relies on high-risk built-in function names

Methodology

Lab Setup

- Virtual Machine 1:
    - OS: Linux
    - Function: Command and Control server

- Virtual Machine 2:
    - OS: Windows 10
    - Function: Logging ETW events during code execution / loading agents

Investigated C2 frameworks

Tested four popular C2 frameworks documented by C2 Matrix project

- Generate .NET agents
- Load .NET assemblies into memory

COVENANT

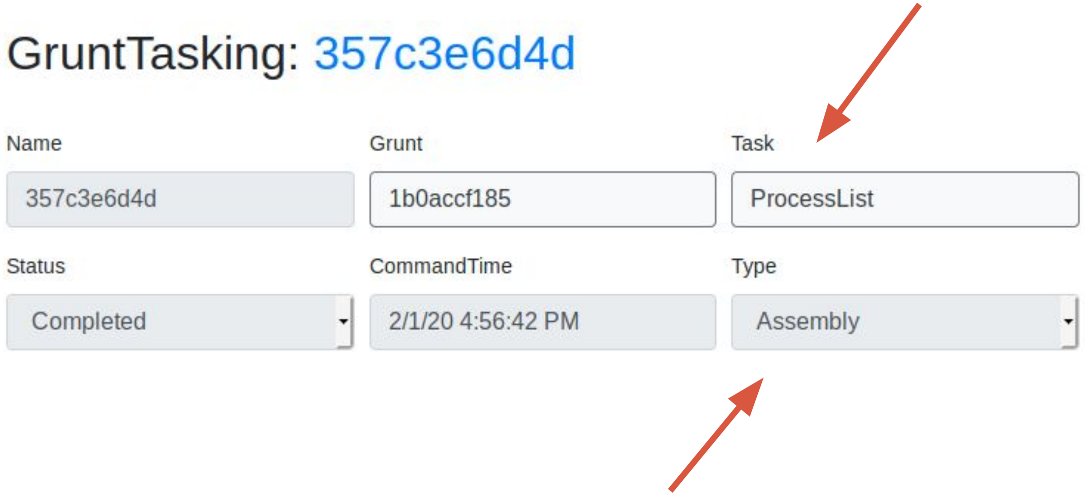
FACTION

SILENTTRINITY

PoshC2

Assembly loading in C2 frameworks

- Executing built-in assembly in Covenant C2



GruntTasking: 357c3e6d4d

Name	Grunt	Task
357c3e6d4d	1b0accf185	ProcessList
Status	CommandTime	Type
Completed	2/1/20 4:56:42 PM	Assembly

Log Creation and Analysis

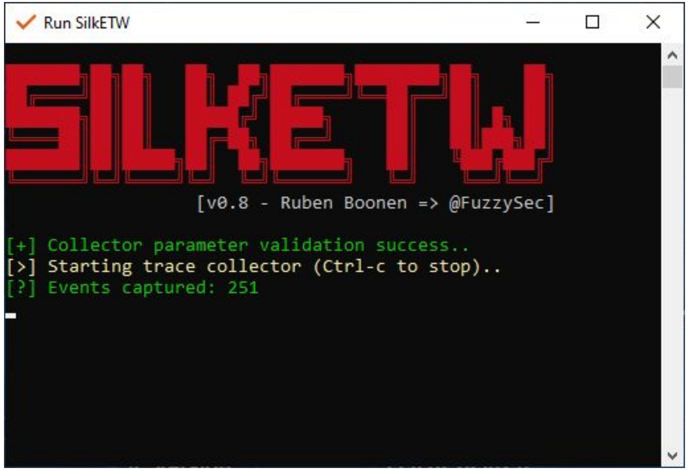
1. Determine relevant ETW providers and event names

2. Generate ETW logs:
   a. Malicious .NET agents
   b. Assembly loading POCs
   c. Benign .NET software

3. Compare event logs side-by-side

SilkETW

- Developed by Ruben Boonen of FireEye
- Logging utility for ETW
- Abstracts complexities
- Entries written to
    - JSON file
    - Windows Event logs
    - Logstash

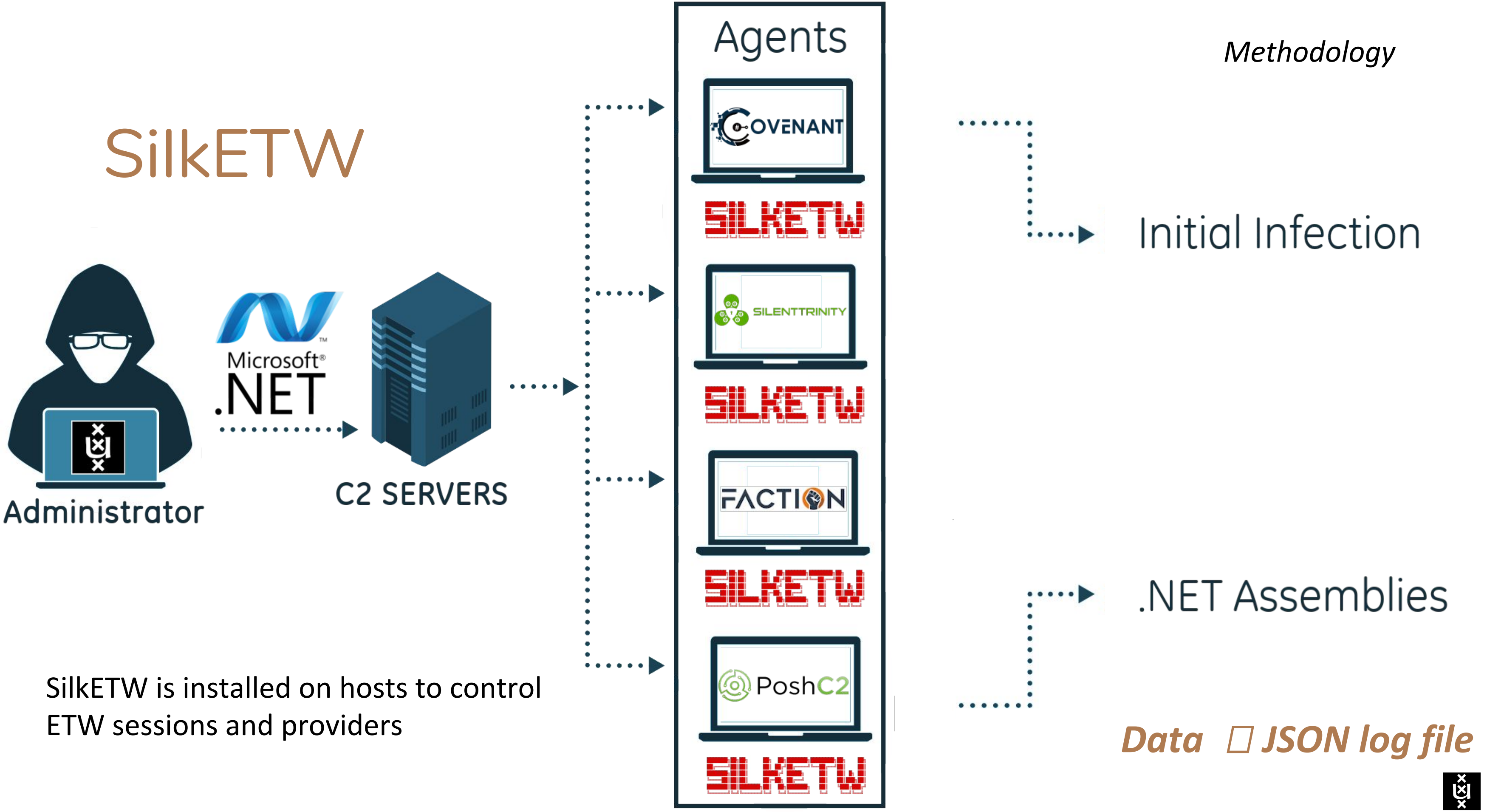


```
[v0.8 - Ruben Boonen => @FuzzySec]

[+] Collector parameter validation success..
[>] Starting trace collector (Ctrl-c to stop)..
[?] Events captured: 251
```

Methodology

SilkETW

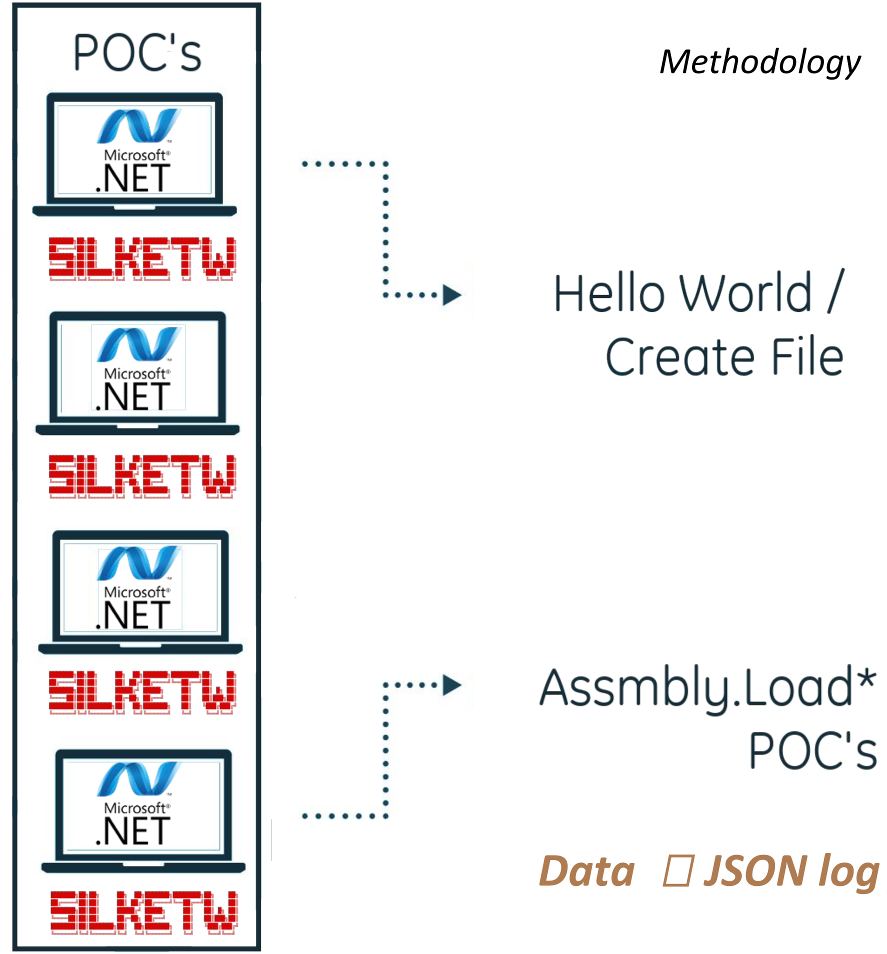


SilkETW is installed on hosts to control ETW sessions and providers

Data □ JSON log file

SilkETW

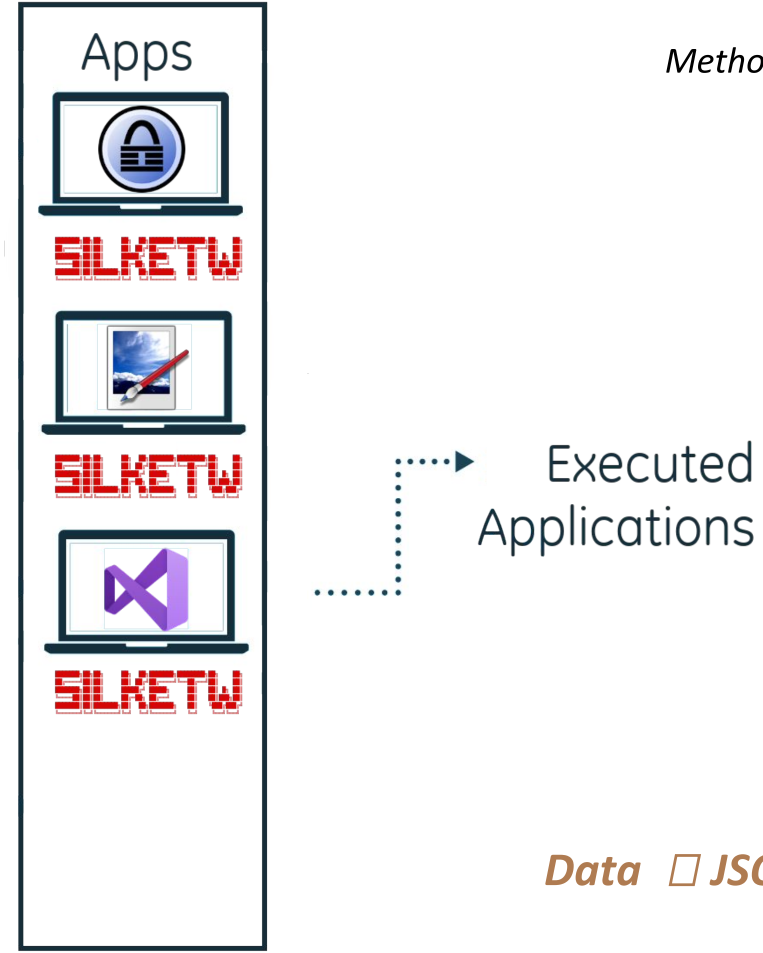
Visual Studio



SilkETW is installed on hosts to control ETW sessions and providers

Data □ JSON log file

SilkETW



SilkETW is installed on hosts to control ETW sessions and providers

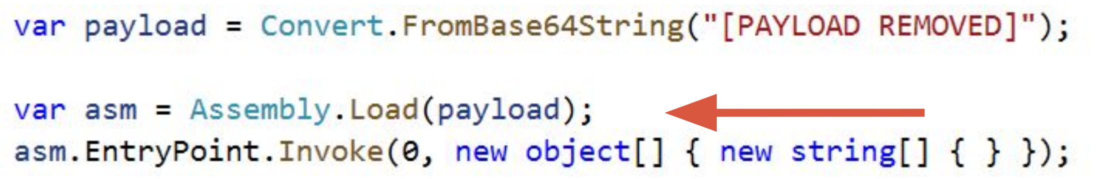
Data □ JSON log file

Example ETW Event (Simplified)

```
{
  "ProviderName": "Microsoft-Windows-DotNETRuntime",
  "EventName": "Loader/AssemblyLoad",
  "TimeStamp": "2020-01-17T07:34:18.0794758-08:00",
  "ProcessName": "N/A",
  ...
  "XmlEventData":{
    "AssemblyFlags": "DomainNeutral|Native",
    "FullyQualifiedAssemblyName": "mscorlib, Version=4.0.0.0, Culture=neutral, PublicKeyToken=...",
    "EventName": "Loader/AssemblyLoad"
    ...
  }
}
```

Results

Assembly.Load



```
var payload = Convert.FromBase64String("[PAYLOAD REMOVED]");

var asm = Assembly.Load(payload);
asm.EntryPoint.Invoke(0, new object[] { new string[] { } });
```

ETW Filtering Steps

Start: Assembly loading POC + logging all .NET-runtime events

99.937 events

26 types of events

Manually clear away irrelevant and verbose event types (Unload, GC, Method/Load, etc.)

End result: Only subscribe to Loader events

9 events

3 types of events

Assembly loading seen from ETW (.NET 4.x)

1. Loader/AssemblyLoad
2. Loader/ModuleLoad
3. Loader/DomainModuleLoad

(* Optional if a module is loaded into an existing assembly)

Assembly loading seen from ETW (.NET 3.5)

1. CLRLoader/ModuleLoad
2. Loader/ModuleLoad

(* Both events contain same information)

Assembly loading seen from ETW

Assembly: Any executable or module, including:

- .NET application itself
- .NET libraries and dependencies
- Dynamically loaded components

AssemblyLoad Event (.NET 4.x)

Legit Module	Assembly name	AssemblyFlags	PublicKeyToken
mscorlib.dll (as observed in Assembly.Load POC)	mscorlib	"DomainNeutral|Native"	**b77a5c561934e089**
mscorlib.dll (as observed in Covenant agent)	mscorlib	"DomainNeutral"	**b77a5c561934e089**

C2 framework	Assembly name	AssemblyFlags	PublicKeyToken
Covenant	"jhyfwkp2.hwm"	"0"	**null**
PoshC2	"Core"	"0"	**null**
FactionC2	"stdlib"	"0"	**null**
SilentTrinity	"Stage"	"Dynamic"	**null**

ModuleLoad Event (.NET 4.x)

Legit Module	ModuleILPath	ModuleNativePath	ModuleFlags
mscorlib.dll (as observed in Assembly.Load POC)	**"C:\\[...]\\mscorlib.dll"**	**"C:\\[...]\\mscorlib.ni.dll"**	"DomainNeutral|Native|Manifest|0x10"
mscorlib.dll (as observed in Covenant agent)	**"C:\\[...]\\mscorlib.dll"**	**""**	"DomainNeutral|Manifest"

C2 framework	ModuleILPath	ModuleNativePath	ModuleFlags
Covenant	**"jhyfwkp2.hwm"**	**""**	"Manifest"
PoshC2	**"Core"**	**""**	"Manifest"
FactionC2	**"stdlib"**	**""**	"Manifest"
SilentTrinity	**"Stage.exe"**	**""**	"Dynamic"

ModuleLoad Event (.NET 3.5)

Legit Module	ModuleILPath	ModuleNativePath	ModuleFlags
mscorlib.dll (as observed in Assembly.Load POC)	**"C:\\[...]\\mscorlib.dll"**	**"C:\\[...]\\mscorlib.ni.dll"**	"3" (DomainNeutral|Native)
mscorlib.dll (as observed in Covenant agent)	**"C:\\[...]\\mscorlib.dll"**	**""**	"1" (DomainNeutral)

C2 framework	ModuleILPath	ModuleNativePath	ModuleFlags
Covenant	**""**	**""**	"0"
FactionC2	**""**	**""**	"0"

ModuleLoad Signature

Field	**Value**
ModuleILPath	No absolute path (i.e. exclude slashes)
ModuleNativePath	Empty string
ModuleFlags (if present)	"0", "Dynamic" or "Manifest"

ModuleLoad Signature - FP Testing

Tested against numerous .NET applications:

- Paint.NET
- KeePass
- Visual Studio

No false positives

Discussion

Limitations - General Considerations

- Assembly loading may occur for legitimate reasons
- Only performed limited false-positive testing
- Different .NET versions result in different event output

Conclusion

Conclusion

How can ETW be leveraged to detect fileless malicious behaviour of .NET agents used by C2 frameworks?

- Agents of multiple C2 frameworks dynamically load assemblies
- Detection possible based on *ModuleLoad* event

Future Work

- Investigate other use cases of ETW for endpoint monitoring
- Investigate real-world implementation of detection

Questions?

Backup slides

Limitations - ModuleLoad signature

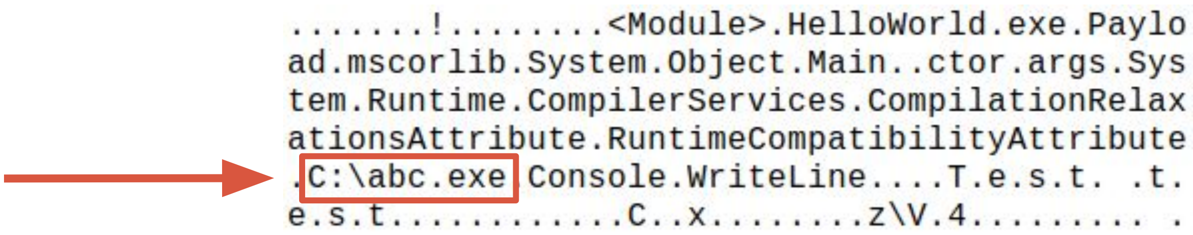
- ModuleLoad signature relies on absence of full path

- Loading assembly file from disk results in absolute path logged in ModuleILPath
    - ***Assembly.LoadFile(string path)***
    - ***Assembly.LoadFrom(string assemblyName)***

Limitations - ModuleLoad signature

- ModuleLoad signature relies on absence of full path

- For dynamically loaded assembly, **ModuleILPath = assembly name**
- Bypass: Patch assembly name with fake path to get fake absolute path logged in ModuleILPath



```
.......!........<Module>.HelloWorld.exe.Paylo
ad.mscorlib.System.Object.Main..ctor.args.Sys
tem.Runtime.CompilerServices.CompilationRelax
ationsAttribute.RuntimeCompatibilityAttribute
.C:\abc.exe.Console.WriteLine....T.e.s.t. .t.
e.s.t...........C..x........z\V.4......... .
```

Documentation

Event	**Field**	**Description**
AssemblyLoad	AssemblyFlags	Type of assembly
	PublicKeyToken	"Last 8 bytes of the SHA-1 hash of the public key under which the application is signed."
ModuleLoad	ModuleILPath	"Path of the Microsoft intermediate language (MSIL) image for the module, or dynamic module name if it is a dynamic assembly."
	ModuleNativePath	"Path of the module native image, if present"
	ModuleFlags	Type of module

Sources:

- https://docs.microsoft.com/en-us/dotnet/framework/performance/loader-etw-events
- https://docs.microsoft.com/en-us/dotnet/api/system.applicationid.publickeytoken

Assembly.Load Variants

Assemblies can be loaded using:

- Assembly.Load
- Assembly.loadFile
- Assembly.LoadFrom
- Assembly.LoadModule
- Assembly.LoadWithPartialName
- Assembly.UnsafeloadFrom